#138 APRIL 1988

2.95 (3.95 CANADA)

Dr. Dobb's Journal of

Software Tools

FOR THE PROFESSIONAL PROGRAMMER

CREATING A NEW AI LANGUAGE

COMBINING RULES AND HYPERTEXT LINKS

REVIEWS

- Turbo Professional 4.0
- Guidelines C++
- Brief 2.0

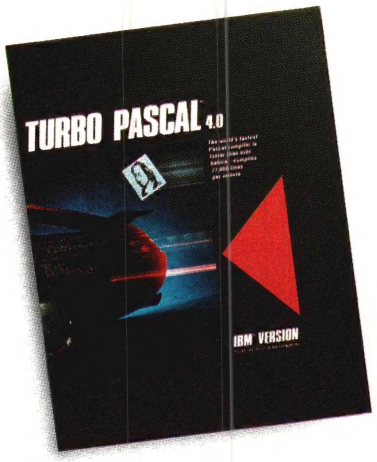
LANGUAGES:

C, Forth, HyperTalk, LISP, Modula-2

0 38351 16562 8 04

DR. DOBB'S JOURNAL #137 APRIL 1988

ter than ever before!



4.0 uses logical units for separate compilation

Pascal 4.0 lets you break up the code gang into "units," or "chunks." These logical modules can be worked with swiftly and separately—so that an error in one module is seeable and fixable, and you're not sent through all your code to find one error. Compiling and linking these separate units happens in a flash because your compiling horsepower is better than 27,000 lines a minute.* And 4.0 also includes an automatic project Make.

4.0's cursor automatically lands on any trouble spot

4.0's interactive error detection and location means that the cursor automatically lands where the error is. While you're compiling or running a program, you get an error message at the top of your screen *and* the cursor flags the error's location for you.

4.0 gives you an integrated programming environment

4.0's integrated environment includes pull-down menus and a built-in editor. Your program output is automatically saved and shown in the output window. You can Scroll, Pan, or Page through all your output and know where everything is all the time. Given 4.0's integration, you can edit, compile, find and correct errors—all from inside the integrated development environment.

You'll never lose your mind, because 4.0 never loses your place

Whenever you re-load 4.0, it remembers what you and it were doing before you left. It puts you right back in the editor with the same file and in the same place as you were working last.

*Run on an 8 MHz IBM AT.

**If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department, and we will arrange a refund.

Please check box(es)	*Sugg. Retail*	*Upgrade Price†*	Serial No.
☐ Turbo Pascal 4.0 Compiler	*$ 99.95*	*$ 39.95*	_____
☐ Turbo Pascal Tutor	*69.95*	*19.95*	_____
☐ Turbo Pascal Database Toolbox	*99.95*	*29.95*	_____
☐ Turbo Pascal Graphix Toolbox	*99.95*	*29.95*	_____
☐ Turbo Pascal Editor Toolbox	*99.95*	*29.95*	_____
☐ Turbo Pascal Numerical Methods Toolbox	*99.95*	*29.95*	_____
☐ Turbo Pascal Gameworks	*99.95*	*29.95*	_____

Total product amount $ _____

CA and MA residents add sales tax $ _____

In US please add $5 shipping and handling for each product ordered $ _____

Outside US please add $10 shipping and handling for each product ordered $ _____

Total amount enclosed $ _____

Please specify diskette size: ☐ 5¼" ☐ 3½"

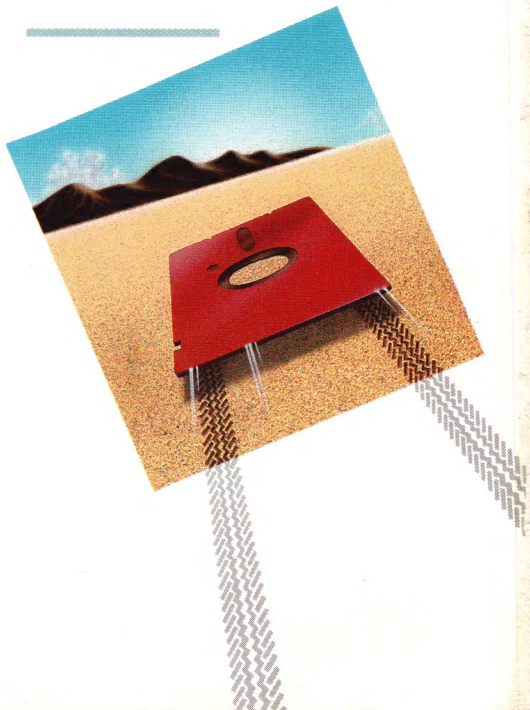
Payment: ☐ VISA ☐ MC ☐ Check ☐ Bank Draft

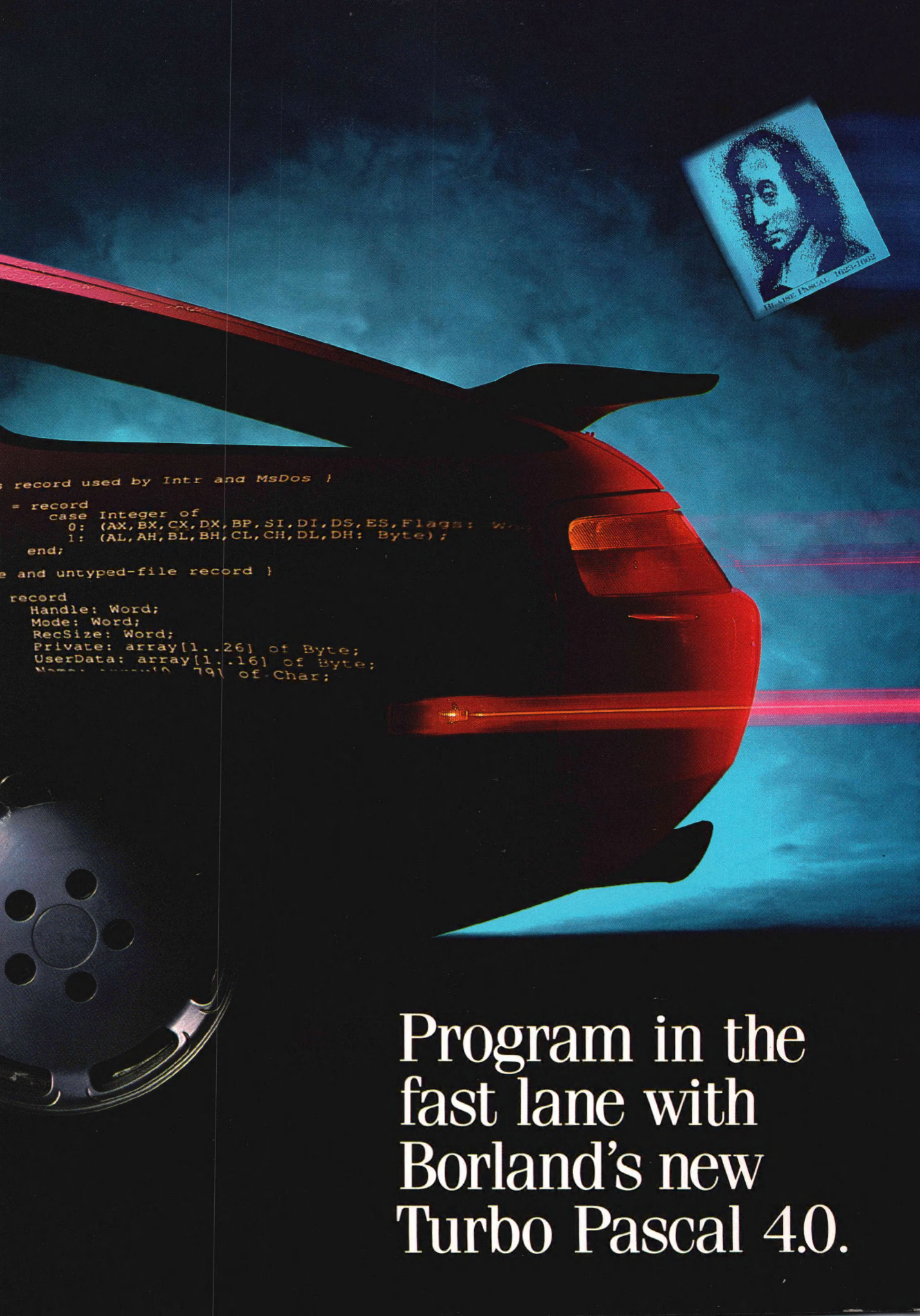
Credit card expiration date: _____/_____

Card # |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

†To qualify for the upgrade price you must give the serial number of the equivalent product you are upgrading.

DDJ 4/88




; record used by Intr and MsDos }
= record
case Integer of
0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:
1: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
end;
e and untyped-file record }
record
Handle: Word;
Mode: Word;
RecSize: Word;
Private: array[1..26] of Byte;
UserData: array[1..16] of Byte;
of Char;
BLAISE PASCAL 1623-1662
Program in the fast lane with Borland's new Turbo Pascal 4.0.

The fast lane is *fas*

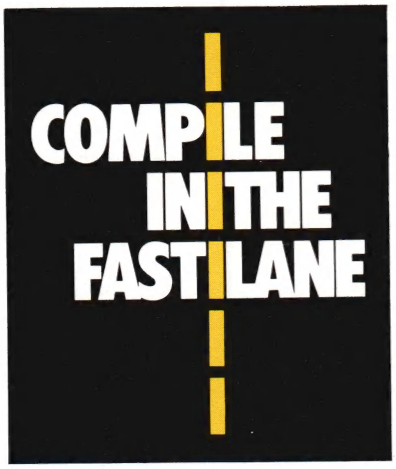
Our new Turbo Pascal® 4.0 is so fast, it's almost reckless. How fast? Better than 27,000 lines of code per minute. That's much faster than 3.0 or any other Pascal compiler and the reason why you need 4.0 today.

Pascal. The fastest and the best.

If you're just now learning a computer language, learn Pascal. If you're already programming in Pascal, you're programming with a winner because Pascal is the worldwide language of choice. Pascal is the most popular language in university computer science classes and with computer enthusiasts who appreciate Pascal's modern programming structure. It's powerful, coherent, easy to learn and use—and with Turbo Pascal 4.0—faster than ever before.

Turbo Pascal: Technical excellence

Commitment to technical excellence and



superiority also means commitment to detail, however painstaking, and that takes time. 4.0's predecessor, Turbo Pascal 3.0 is the worldwide standard, and with Turbo Pascal 4.0, we've bettered that standard. 4.0 is clearly the world's fastest development tool for the IBM® PS/2 series, PC's and compatibles—and the world's favorite Pascal compiler.

4.0 breaks the code barrier

No more swapping code in and out to beat the 64K code barrier. Designed for large programs, Turbo Pascal 4.0 lets you use all 640K memory in your computer. You paid for all that memory, now you can use it freely.

For the IBM PS/2 and the IBM and Compaq families of personal computers and all 100% compatibles.

YES!

I want to upgrade to Turbo Pascal 4.0 and the 4.0 Toolboxes

Registered owners have been notified by mail. If you are a registered Turbo Pascal user and have not been notified of Version 4.0 by mail, please call us at *(800) 543-7543*. To upgrade if you have not registered your product, just send the original registration form from your manual and payment with this completed coupon to:

Pascal 4.0 Upgrade Dept.
Borland International
4585 Scotts Valley Drive
Scotts Valley, CA 95066

Name _____

Ship Address _____

City _____ State _____

Zip _____ Telephone ( ) _____

This offer is limited to one upgrade per valid registered product. It is good until June 30, 1988. Not good with any other offer from Borland.
Outside U.S. make payments by bank draft payable in U.S. dollars drawn on a U.S. bank. CODs and purchase orders will not be accepted by Borland.

DDJ 4/88

Now's the time for a *fast* decision: Upgrade now to 4.0!

Compatibility with Turbo Pascal 3.0

We've created 4.0 to be highly compatible with version 3.0 and included a conversion program and compatibility units to help you convert all your 3.0 programs to 4.0.

Highlights of Borland's new Turbo Pascal 4.0

- Compiles 27,000 lines per minute
- Supports >64K programs
- Uses units for separate compilation
- Integrated development environment
- Interactive error detection/location
- Includes a command line version of the compiler

4.0 also

- Saves output screen in a window
- Supports 25, 43 and 50 lines per screen
- Generates MAP files for debugging
- Has graph units including CGA, EGA, VGA, MCGA, 3270 PC, AT & T 6300 & Hercules support
- Supports extended data types (including word, long integers)
- Does smart linking
- Comes with a free revised MicroCalc spreadsheet source code

4.0 is all yours for only $99.95

Sieve (25 iterations)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Size of Executable File	2224 bytes	11682 bytes
Execution speed	9.3 seconds	9.7 seconds

Sieve of Eratosthenes, run on an 8MHz IBM AT

Since the source file above is too small to indicate a difference in compilation speed we compiled our GOMOKU program from Turbo Gameworks to give you a true sense of how much faster 4.0 really is!

Compilation of GO.PAS (1006 lines)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Compilation speed	2.2 seconds	3.6 seconds
Lines per minute	27,436	16,750

GO.PAS compiled on an 8 MHz IBM AT

60-Day Money-Back Guarantee**

BORLAND INTERNATIONAL

For the dealer nearest you or to order call

(800) 543-7543.

CIRCLE NO. 101 ON READER SERVICE CARD

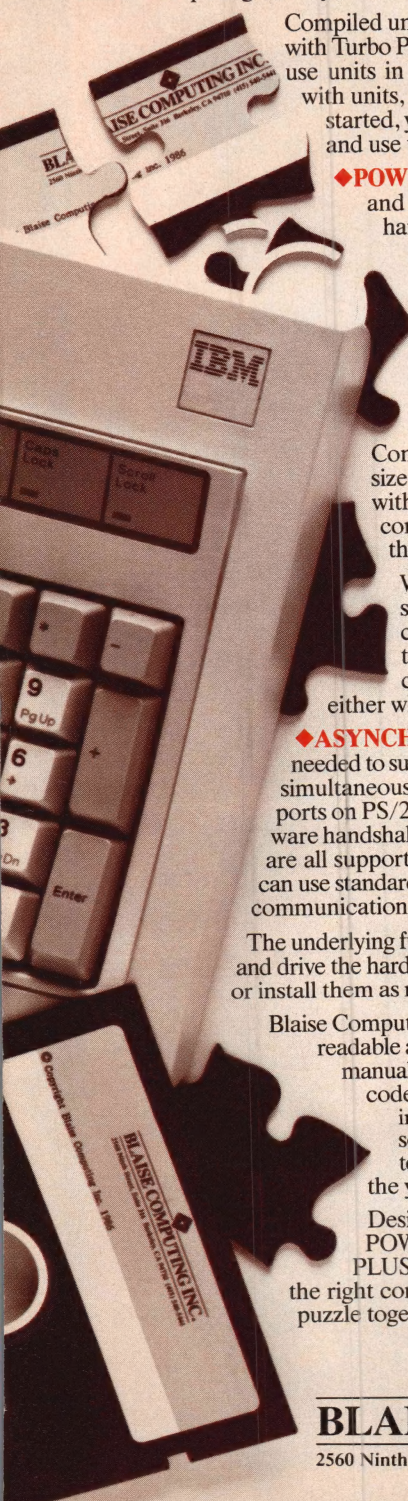
BORLAND

Interlocking Pieces:

Blaise and Turbo Pascal.

Now, for Turbo Pascal 4.0!

Whether you're a Turbo Pascal expert or a novice, you can benefit from using professional tools to enhance your programs. With Turbo POWER TOOLS PLUS™ and Turbo ASYNCH PLUS,™ Blaise Computing offers you all the right pieces to solve your 4.0 development puzzle.



Compiled units (TPU files) are provided so each package is ready to use with Turbo Pascal 4.0. Both POWER TOOLS PLUS and ASYNCH PLUS use units in a clear, consistent and effective way. If you are familiar with units, you will appreciate the organization. If you are just getting started, you will find the approach an illustration of how to construct and use units.

◆**POWER TOOLS PLUS** is a library of over 180 powerful functions and procedures like fast direct video access, general screen handling including multiple monitors, VGA and EGA 50-line and 43-line text mode, and full keyboard support, including the 101/102-key keyboard. Stackable and removable windows with optional borders, titles and cursor memory provide complete windowing capabilities. Horizontal, vertical, grid and Lotus-style menus can be easily incorporated into your programs using the menu management routines. You can create the same kind of moving pull down menus that Turbo Pascal 4.0 uses.

Control DOS memory allocation. Alter the Turbo Pascal heap size when your program executes. Execute any program from within your program and POWER TOOLS PLUS automatically compresses your heap memory if necessary. You can even force the output of the program into a window!

Write general interrupt service routines for either hardware or software interrupts. Blaise Computing's unique intervention code lets you develop memory resident (TSRs) applications that take full advantage of DOS capabilities. With simple procedure calls, "schedule" a Turbo Pascal procedure to execute either when pressing a "hot key" or at a specified time.

◆**ASYNCH PLUS** provides the crucial core of hardware interrupts needed to support asynchronous data communications. This package offers simultaneous buffered input and output to both COM ports, and up to four ports on PS/2 systems. Speeds to 19.2K baud, XON/XOFF protocol, hardware handshaking, XMODEM (with CRC) file transfer and modem control are all supported. ASYNCH PLUS provides text file device drivers so you can use standard "Readln" and "Writeln" calls and still exploit interrupt-driven communication.

The underlying functions of ASYNCH PLUS are carefully crafted in assembler and drive the hardware directly. Link these functions directly to your application or install them as memory resident.

Blaise Computing products include all source code that is efficiently crafted, readable and easy to modify. Accompanying each package is an indexed manual describing each procedure and function in detail with example code fragments. Many complete examples and useful utilities are included on the diskettes. The documentation, examples and source code reflect the attention to detail and commitment to technical support that have distinguished Blaise Computing over the years.

Designed explicitly for Turbo Pascal 4.0, Turbo POWER TOOLS PLUS and Turbo ASYNCH PLUS provide reliable, fast, professional routines—the right combination of pieces to put your Turbo Pascal puzzle together. **Complete price is $129.00 each.**

THE BLAISE M E N U

Turbo POWER SCREEN *$129.00*
NEW! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. Now for Turbo Pascal 4.0, soon for C and BASIC.

Turbo C TOOLS *$129.00*
Full spectrum of general service utility functions including: windows; menus; memory resident applications; interrupt service routines; intervention code; and direct video access for fast screen handling. For Turbo C.

C TOOLS PLUS *$129.00*
Windows; menus; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. Specifically designed for Microsoft C 5.0 and QuickC.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM file transfer. For Microsoft C and Turbo C or MS Pascal.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

KeyPilot *$49.95*
"Super-batch" program. Create batch files which can invoke programs and provide input to them; run any program unattended; create demonstration programs; analyze keyboard usage.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

RUNOFF *$49.95*
Text formatter for all programmers. Written in Turbo Pascal; flexible printer control; user-defined variables; index generation; and a general macro facility.

TO ORDER CALL TOLL FREE
800-333-8087

TELEX NUMBER - 338139

YES! Send me the right pieces!
Enclosed is $_____ for _____ copies of _____.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name:_____ Phone: (_____) _____-_____
Address:_____ State:_____ Zip:_____
City:_____ Exp. Date:_____
VISA or MC#:_____

Microsoft and QuickC are registered trademarks of Microsoft Corporation. Turbo Pascal is a registered trademark of Borland International.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

CIRCLE NO. 102 ON READER SERVICE CARD

CONTENTS

ARTICLES

REVIEWS

COLUMNS

FORUM

PROGRAMMER'S SERVICES

About the Cover
They said robots were an AI cliche, but we replied, "Hey, we may be April fools, but we're not dummies!" Special thanks to Android Studios, of Oakland, Calif. for central casting, and to Lord Whorfin for the insulating fluid.

Next Issue
May will see a number of changes in *DDJ*, including a new column on "Programming Paradigms." The issue focus is on developing applications from a number of perspectives, as well as the usual surprises.

PROGRAMMER'S PARADISE PRESENTS

SoftCode- The Powerful New Program Generator from THE SOFTWARE BOTTLING COMPANY

SoftCode — Available exclusively from Programmer's Paradise and the Software Bottling Company.

SoftCode is a new program generator that's based on the idea that you should never have to write the same program twice. And that's no small idea! Because if you're like most programmers, 75% of your time is spent rewriting and modifying the same old code to go into new programs.

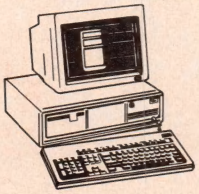
SoftCode, the most powerful screen editor available today, is packed with features that deliver fast results. Draw boxes, change colors, move blocks around, place fields and more. Then let SoftCode generate your program ... Automatically.

The unique template language allows you to instruct SoftCode to generate any kind of program in your own personalized style and in your own language.

To get started, SoftCode includes a set of pre-written templates for complete data entry routines. You choose the language: BASIC, C, Pascal, or dBASE.

Additional Templates are available at the Programmer's Paradise low price of $45.00.
List $129 Introductory Price $109

Flash-Up



Flash-Up enhances every C program ... without writing a single line of code. Just use the RAM resident window editor to "draw" menus and help windows into any program.

Think of Flash-Up as a full keyboard macro utility, a help note annotator and a menu maker rolled into one to help you do things that could never be done before. Like making smart menus that send keyboard macros. And automatically attaching them to your programs with "electronic glue" and even adding mouse support to all your programs.
List $89 Special $75

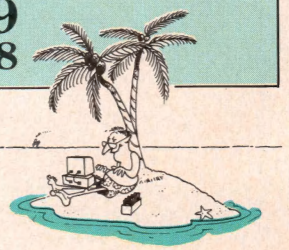
Flash-Up Toolbox

FREE

Also try Flash-Up Developers Toolbox. You get the Programming Language Interface and a royalty-free routine module. Imagine controlling keyboard macros and smart windows by sending commands from any programming language! And including them free in every program you distribute.
List $49 Special $45
List $49 Special FREE
(with purchase of Flash-Up)

See why Flash-Up and the Toolbox were nominated for PC Magazine's Technical Excellence award for 1987/1988.

1-800-445-7899
In NY: 914-332-4548



Programmer's Paradise™
A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591

CIRCLE NO. 103 ON READER SERVICE CARD

Call or write for the latest catalog

DISCOVER PARADISE

Programmer's Paradise Gives You Superb Selection, Personal Service and Unbeatable Prices!

Welcome to Paradise. The microcomputer software source that caters to your programming needs.

Discover the Many Advantages of Paradise...

- Lowest price guaranteed
- Latest versions
- Huge inventory, immediate shipment
- Knowledgeable sales staff
- Special orders
- 30-day money-back guarantee

Over 500 brand-name products in stock—if you don't see it, call!

We'll Match Any Nationally Advertised Price.

		LIST	OURS
386 SOFTWARE			
386-TO-THE-MAX	*NEW*	75	65
ADVANTAGE 386 C OR PASCAL		895	829
DESQVIEW		130	109
FOXBASE + 386		595	CALL
HIGH C-386	*NEW*	895	835
MICROPORT SYSTEM V/386 (COMPLETE)		799	679
MS WINDOWS/386	*SPECIAL*	195	125
NDP C OR FORTRAN-386	*NEW*	595	545
PHARLAP 386/ASM/LINK		495	419
SCO XENIX SYS V 386 (COMPLETE)		1495	1195
VM/386	*SPECIAL*	245	179
X-AM		595	535
ARTIFICIAL INTELLIGENCE			
ARITY STANDARD PROLOG		95	79
MULISP-87 INTERPRETER		300	199
PC SCHEME		95	85
TURBO PROLOG		100	69
TURBO PROLOG TOOLBOX		100	69
ASSEMBLERS/LINKERS			
ADVANTAGE DISASSEMBLER		295	279
MS MACRO ASSEMBLER		150	99
OPTASM	*NEW*	195	169
PASM86		195	115
PLINK86PLUS		495	279
BASIC			
DB/LIB		139	119
FINALLY!		99	89
FLASH-UP	*SPECIAL*	89	75
MACH 2		75	59
MS QUICKBASIC		99	65
QUICKPAK		69	59
QUICKWINDOWS W/SOURCE		99	89
TRUE BASIC		100	79
TURBO BASIC		100	69
TURBO BASIC TOOLBOXES		100	69

MOUSE PRODUCTS

		LIST	OURS
LOGITECH SERIAL OR BUS MOUSE W/PLUS, SOFTWARE		119	99
W/PLUS, LOGICPAINT		149	119
W/PLUS, LOGICADD MOUSE		189	149
W/PLUS, PUBLISHER MOUSE		179	145
W/PLUS, PAINT, CADD		199	159
W/PLUS, CADD PUBL. MOUSE		239	189
W/PLUS, PAINT, PUBL. MOUSE		199	159
W/PLUS, PAINT, CADD, PUBL.		253	205
LOGITECH SERIES 2 W/PLUS		99	79
MICROSOFT SER OR BUS MOUSE		150	99
W/EASY CAD		175	119
W/MS WINDOWS		200	139
PC MOUSE BUS W/PNT & POPUPS		179	129
PC MOUSE SER. W/PNT & POPUPS		159	115
SUMMAMOUSE		119	99

		LIST	OURS
C COMPILERS			
LATTICE C		500	269
MICROSOFT C		450	285
QUICK C	*SPECIAL*	99	65
TURBO C		100	65
C INTERPRETERS			
C-TERP		298	229
INSTANT C		495	379
RUN/C		120	79
RUN/C PROFESSIONAL		250	155

		LIST	OURS
C LIBRARIES			
C ASYNCH MANAGER		175	135
C-FOOD SMORGASBORD		150	95
C TOOLS PLUS/5.0		129	99
C UTILITY LIBRARY		185	125
ESSENTIAL COMMUNICATIONS		185	125
COMMUNICATIONS PLUS		250	189
GREENLEAF C SAMPLER		95	69
GREENLEAF COMM LIBRARY		185	125
GREENLEAF FUNCTIONS		185	125
MULTI-C		149	135
PFORCE		295	215
RESIDENT C W/SOURCE		198	169
TIMESLICER	*SPECIAL*	295	269
TURBO C TOOLS		129	99
COBOL			
E-Z PAGE		295	259
MICRO FOCUS COBOL/2		900	729
COBOL/2 TLSET		900	729
PC-CICS		1500	1189
LEVEL II COBOL		349	279
PERSONAL COBOL		149	119
OTHERS		CALL	CALL
MICROSOFT COBOL		700	449
MICROSOFT SORT		195	129
OPT-TECH SORT		149	105
REALCICS		995	785
REALIA COBOL W/REALMENU		995	785
RM/COBOL		1145	899
RM/COBOL-85		950	759
RM/SCREENS		1250	999
SCREENIO		395	315
		400	379
COMMUNICATIONS			
ASCOM IV		195	175
CARBON COPY PLUS		195	159
CO-SESSION (2 USER)	*NEW*	249	225
SUPPORT	*NEW*	175	155
APPLICATION	*NEW*	125	115
PTEL		50	45
SIDETALK		120	89
DEBUGGERS			
ADVANCED TRACE-86		175	119
PERISCOPE I	*SPECIAL*	345	269
PERISCOPE II	*SPECIAL*	175	135
PERISCOPE III 8 MHZ	*SPECIAL*	995	789
PERISCOPE III 10 MHZ	*SPECIAL*	1095	869
PFIX 86 PLUS	*SPECIAL*	395	199
DISK/DOS/KEYBOARD UTILITIES			
ADVANCED NORTON UTILITIES		150	99
COMMAND PLUS	*NEW* V. 2.0	80	69
DISK OPTIMIZER		70	65
FETCH		55	45
NORTON COMMANDER		75	55
PC TOOLS DELUXE	*NEW*	80	69
PDISK		145	105
VFEATURE		80	75
EDITORS			
BRIEF		195	CALL
W/DBRIEF		275	CALL
EMACS		295	265
EPSILON		195	149
KEDIT		125	99
MKS Vi		75	69
MULTI-EDIT	*NEW*	99	89
PC/EDT		250	229
PMATE		195	115
SPF/PC		245	185
VEDIT PLUS		185	129

		LIST	OURS
FILE MANAGEMENT			
BTRIEVE		245	185
XTRIEVE		245	189
REPORT OPTION		145	109
BTRIEVE/N		595	455
XTRIEVE/N		595	459
REPORT OPTION/N		345	269
CBTREE		159	139
C-TREE		395	315
R-TREE		295	239
C-TREE/R-TREE BUNDLE		650	519
D-TREE	*NEW*	395	CALL
DBC III		250	169
DBC III PLUS		750	595
DB_VISTA OR DB_QUERY		195	CALL
SINGLE USER W/SOURCE		495	CALL
MULTIUSER		495	CALL
MULTIUSER W/SOURCE		990	CALL
INFORMIX PRODUCTS		CALL	CALL
PHACT MANAGER		249	229
XQL		795	599
FORTRAN COMPILERS			
LAHEY FORTRAN F77L EM/16		695	625
LAHEY PERSONAL FORTRAN 77		95	89
MICROSOFT FORTRAN		450	285
RM/FORTRAN		595	479
FORTRAN LIBRARIES/UTILITIES			
DIAGRAM'ER OR DOCUMENT'ER		129	115
GRAFMATIC OR PLOTMATIC		135	119
MAGUS NUMERICAL ANALYST		295	249
MATHPAC		495	445
SPINDRIFT LIBRARY	*SPECIAL*	149	129
SSP/PC		350	269
GRAPHICS			
ADVANTAGE GRAPHICS (C)		250	229
ESSENTIAL GRAPHICS		250	189
GSS GRAPHIC DEV. TOOLKIT		495	375
HALO		300	209
HALO (5 MICROSOFT LANG.)		595	399
METAWINDOW PLUS		275	229
TURBOWINDOW/C		95	79
TURBO HALO (FOR TURBO C)		99	79
MODULA-2			
LOGITECH MODULA-2 COMPILER KIT		99	79
DEVELOPMENT SYSTEM		249	199
TOOLKIT		169	139
SOLID B + TOOLBOX	*NEW*	99	89
STONYBROOK MODULA-2 W/UTILITIES		195	169
		345	299
OBJECT-ORIENTED PROGRAMMING			
ACTOR		495	419
ADVANTAGE C++		495	479
PFORCE++		395	215
SMALLTALK/V		100	85
SMALLTALK/V286	*NEW*	200	169
APPLICATION PACKS		50	45
OPERATING SYSTEMS			
MICROPORT SYS V/AT		549	469
SCO XENIX SYSTEM V		1295	995
WENDIN-DOS		99	79
OTHER MICROPORT, SCO, WENDIN PRODUCTS		CALL	CALL
PASCAL COMPILERS			
MICROSOFT PASCAL		300	189
PASCAL-2		259	CALL
TURBO PASCAL		100	69
TURBO PASCAL DEV.LIB.		395	289
BORLAND ADD-ONS		CALL	CALL

		LIST	OURS
TURBO PASCAL ADD-ONS			
ASCII TURBO GHOST WRITER STARTER	*NEW*	99	89
COMPLETE	*NEW*	289	259

FEATURED PRODUCTS

NDP FORTRAN-386—Fast globally optimizing compiler for the 80386. Use up to 4 gigabytes per unit to generate programs, procedures and arrays. A full implementation of FORTRAN-77 with popular extensions. Output is assembly language. (Requires PharLap ASM/LINK)
List: $595 **Special Price: $545**

SPINDRIFT LIBRARY—Collection of 150 callable subroutines for Fortran, that provides access to WINDOWS, DOS KEYBOARD, and SCREEN. The windowing system allows an unlimited number of windows to be defined on the screen, each having its own color, cursor position and border type. Supports automatic wrap and/or scrolling.
List: $149 **Special Price: $129**

PANEL PLUS—Screen management library for data display, entry, and editing. Supports pop-up fields and windows, multi-line fields, horizontal and vertical field scrolling, menus, help boxes, and custom field validation.
List: $495 **Special Price: $389**

MKS RCS—Revision Control System manages multiple revisions of text files and program sources code on DOS systems. Storage, retrieval, tracking and branching text files are effectively managed.
List: $189 **Intro Price: $169**

		LIST	OURS
AZATAR DOS TOOLKIT	*NEW*	99	85
DOS/BIOS & MOUSE TOOLS		75	69
FLASH-UP		89	79
METRABYTE DATA ACQ. TOOLS		100	89
SCREEN SCULPTOR		125	95
SYSTEM BUILDER		150	129
IMPEX		100	89
REPORT BUILDER		130	115
T-DEBUG PLUS V. 4.0	*SPECIAL*	45	39
W/SOURCE	*NEW*	90	79
TURBO.ASM		99	69
TURBO ASYNCH PLUS		129	99
TURBO GEOMETRY LIBRARY	*NEW*	100	89
TURBO HALO		99	85
TURBO MAGIC		99	89
TURBO POWER TOOLS PLUS		129	99
TURBO POWER UTILITIES		95	79
TURBO PROFESSIONAL 4.0		99	79
TURBO WINDOW/PASCAL		95	79
SCREENS/WINDOWS			
C-SCAPE		279	265
CURSES W/SOURCE CODE		250	169
GREENLEAF DATA WINDOWS		225	155
W/SOURCE CODE		395	259
HI-SCREEN XL		149	119
JYACC FORMAKER		495	449
JYACC JAM		750	679
MICROSOFT WINDOWS		99	65
MS WINDOWS DEVELOPMENT KIT		500	319
PANEL PLUS	*SPECIAL*	495	389
PANEL/QC OR /TC		129	99
SCREENSTAR W/SOURCE		198	169
VITAMIN C	*SPECIAL*	225	149
VC SCREEN		99	79
VIEW MANAGER		275	199
WINDOWS FOR DATA		295	CALL
W/SOURCE		590	CALL
ADDITIONAL PRODUCTS			
ADVANTAGE VCMS		379	329
BASTOC		495	399

XENIX/UNIX SOFTWARE

	LIST	OURS
MICROPORT & SCO PRODUCTS	CALL	CALL
ADVANTAGE C++	695	625
BTRIEVE/N	595	455
DIRECTORY SHELL (286)	349	315
DIRECTORY SHELL (386)	495	445
EPSILON	195	149
FOXBASE +	795	729
INFORMIX PRODUCTS	CALL	CALL
JYACC FORMAKER	895	809
JYACC JAM	1350	1219
KORN SHELL	145	115
MICROSOFT LANGUAGES	CALL	CALL
PANEL PLUS	795	675
RM/COBOL	1250	949
RM/FORTRAN	750	549
WINDOWS FOR DATA	795	CALL

		LIST	OURS
DAN BRICKLIN'S DEMO PROGRAM		75	59
DEMO PROGRAM II		195	155
DB2C		299	CALL
FLOW CHARTING II		229	205
MAGIC PC		195	179
MKS TOOLKIT		139	115
MKS RCS	*NEW, SPECIAL*	189	169
MKS-SQPS	*NEW*	495	469
NORTON GUIDES		100	65
PC-LINT	*SPECIAL*	139	95
POLYMAKE		149	129
POLYTRON PVCS		CALL	CALL
PRE-C		295	159
SOURCE PRINT		95	75
TREE DIAGRAMMER		77	69

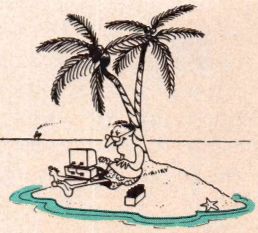
Terms and Policies
- **We honor MC, VISA, AMERICAN EXPRESS**

No surcharge on credit card or C.O.D. Prepayment by check. New York State residents add applicable sales tax. Shipping and handling $3.95 per item, sent UPS ground. Rush service available, prevailing rates.
- Programmer's Paradise will match any current nationally advertised price for the products listed in this ad.
- Prices and Policies subject to change without notice.
- Hours 9AM EST—7PM EST
- We'll Match any Nationally Advertised Price
- ***Mail Orders include your phone number***
- *Ask for details. Some manufacturers will not allow returns once disk seals are broken.

Dealers and Corporate Buyers—Call for special discounts and benefits!

1-800-445-7899
In NY: 914-332-4548
Customer Service: 914-332-0869
International Orders: 914-332-4548
Telex: 510-601-7602

Programmer's Paradise™



A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591

CIRCLE NO. 104 ON READER SERVICE CARD

EDITORIAL

Sizing Up Microsoft

It's been said from time to time that *Dr. Dobb's* has a love/hate relationship with a little software company up in Washington state, a company you may have heard of. Some people who read the magazine often think we're biased in the company's favor because we mention it so often, or because we haven't published a scathing indictment of the bugs in one of their compilers (lately). Other people, who presumably don't talk to the folks in the first camp, charge that we spend too much time covering Microsoftian developments and urge that we spend more time covering (fill in the name of your favorite product category).

While there seems to be almost a tradition of Microsoft "bashing" in the industry, that's not what's going on here. What's really going on is that the company has grown so much in the last few years that we're all becoming victims of shifting paradigms. Although it may be hard for us (especially the old-timers) to appreciate, there's no escaping it: If you're a serious developer in the personal computer marketplace, you have to analyze Microsoft's products and strategies and how that impacts your products and plans. Often that means reading both for what was said, and what wasn't; analyzing both for tactics and strategy.

Which brings us to the point of this little essay: For most developers, Microsoft's role can be seen today as primarily of strategic rather than tactical importance. Increasingly, the details of specific announcements are becoming less important than strategic concerns.

Case in point: OS/2. It took years to develop—big companies move slowly—and it'll take years for the new generation of applications to appear. But despite the tactical opportunity for other operating systems to penetrate the market (in relatively small numbers), the strategic reality is that OS/2 will undoubtably prevail in the coming years unless IBM pulls the plug. OS/2's victory will have little to do with technical merit, and everything to do with the incredible resources of Microsoft and IBM. OS/2 is the Ada of personal computer operating systems: less attractive than other options, but massive and impossible to assail.

In reflecting on this state of affairs you might be disappointed, like many of us, that Microsoft's strategy ignores the 386 market. It is ironic that most of the scheduled acceptance of OS/2 will occur in a year or more, just when (rumors say) we'll all be hip-deep in 386 machines, but this just further emphasizes the strategic choices developers have. You can take the safe route and write applications for OS/2, or consider all those 386 machines crying for new software. A great deal of successful 386 software is being written by small companies, by people who are able to respond quickly to changing market conditions.

And finally, there are times when it's to everyone's benefit to have a big company leading the way. Microsoft's involvement in CD-ROM is a perfect example. The company's extensions for MS-DOS are an important tool for everyone, but the most important concern right now is making CD-ROM viable. In a market deadlocked with the hardware companies are waiting for the software companies and vice versa, Microsoft has taken the laudable step of continually pushing ahead with CD-ROM products and support. Which means we all might get to reap the benefits of CD-ROM before the turn of the century—presumably even those of us running MS-DOS as a task under a 386 version of Unix.

Tyler Sperry

Tyler Sperry
editor

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Tyler Sperry*
Managing Editor *Monica E. Berg*
Associate Editor *Ron Copeland*
Assistant Editor *Sara Noah Ruddy*
Technical Editors *Allen Holub, Richard Relph, Kent Porter*
Contributing Editors *Stan Krute, Martin Tracy*
Copy Editor *Rhoda Simmons*
Art/Production
Director *Larry L. Clay*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Barbara Mautz*
Typesetter *Mary E. Lopez*
Cover Photographer *Michael Carr*
Circulation
Circulation Director *Maureen Kaminski*
Fulfillment Coordinator *Francesca Martin*
Subscription Supervisor *Kathleen Shay*
Newsstand Coordinator *Sarah Frisbie*
Administration
Vice President of Finance and Operations *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura DiLazzaro*
Marketing/Advertising
Director *Ferris Ferdon*
Advertising Coordinator *Patricia Albert*
Account Managers *see page 129*

Publisher
Peter Hutchinson

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly, with two special issues per year by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Associate Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Request: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 3713, Escondido, CA 92025. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $28 per year airmail or $11 per year surface. All other countries add $32 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX 620430 (WUI).

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C. F. von Quadt*
President *Laird Foshay*
Vice President of Publishing *William P. Howard*

Aztec C

Power to go the distance... Whatever that distance might be

From real time embedded applications to comprehensive commercial applications on Macintosh, IBM PC, Amiga, Atari, and others, Aztec C has earned a well-deserved reputation as an innovative, tough to beat, rock-solid C development system.

But don't just take our word for it—try it yourself. We know that the best way to understand what puts you ahead with Aztec C is to use it. That's why Aztec C systems purchased directly from Manx come with a 30-day, no questions asked, satisfaction guarantee. Call for yours today.

We can also send you information that details the special features and options of Aztec C. Plus information on support software, extended technical support options, and all of the services and specialized support that you may need when you're pushing your software to the limits and... beyond.

MS-DOS Hosted ROM Development Systems

Host + Target: $750 Additional Targets: $500

Targets:
- **6502 family**
- **8080-8085-Z80-Z180-64180**
- **8088-8086-80186-80286/8087-80287**
- **68000-68010-68020/68881**

Components:
- **C compiler for host and target**
- **Assembler for host and target**
- **linker and librarian**
- **Unix utilities make, diff, grep**
- **Unix vi editor**
- **debugger**
- **download support**

Features:
- **Complete development system**
- **Fast development times**
- **Prototype and debug non-specific code under MS-DOS**
- **Compilers produce modifiable assembler output, support inline assembly, and will link with assembly modules**
- **Support for INTEL hex, S record, and other formats**
- **source for UNIX run time library**
- **processor dependent features**
- **source for startup**

Aztec C Micro Systems

Aztec C is available for most microcomputers in three configurations: The Professional; The Developer; and The Commercial system. All systems are upgradable.

Aztec C68k/Am .... Amiga
source debugger–optional

Aztec C68k/Mac ... Macintosh
MPW and MAC II support

Aztec C86 ..... MS-DOS
source debugger • CP/M libraries

The following have special pricing and configurations. Call for details.

Aztec C68k/At ..... Atari ST
Aztec C80 ..... CP/M-80
Aztec C65 ..... Apple II & II GS

Standard System $199
- C compiler
- Macro Assembler
- overlay linker with librarian
- debugger
- UNIX and other libraries
- utilities

Developer System $299
- all Standard System features
- UNIX utilities make, diff, grep
- UNIX vi editor

Commercial System $499
- all Developer features
- source for run time libraries
- one year of updates

MANX

C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724

Aztec C is available on a thirty-day money back guarantee. Call now and find out why over 50,000 users give Aztec C one of the highest user-satisfaction ratings in the industry.

Call 1-800-221-0440

In NJ or outside the USA, call 201-542-2121
Telex: 4995812 Fax 201-542-8386

CIRCLE NO. 105 ON READER SERVICE CARD

RUNNING LIGHT

As my friend Ezra Shapiro has pointed out, when it comes to Artificial Intelligence in personal computer software, we've still got a long way to go. For all our simulations of neural nets and expert systems, our machines still seem woefully inadequate in reasoning power. Today someone told me about an advanced communications package that recognizes when it's been invoked without being configured and actually prompts the user for the information it needs. While I acted suitably impressed, I couldn't help wondering why the program couldn't check the hardware, access a database containing parameters and phone numbers, and furnish the user with a list of assumptions for approval. It could be that I just expect too much; the program my friend described is apparently considered a fairly sophisticated program by his peer group.

Yes, Ezra, still we have a long way to go.

DDJ readers, as a class, probably have less road to travel than other programmers. This issue, after all, has a couple of articles designed to stimulate your interest in expanding what we commonly describe as AI languages. As we investigated articles for this issue, I discovered that for many people the line between AI languages and "ordinary" languages has blurred. For Bill and Bev Thompson, authors of "Topics in Knowledge-Based Languages," (page 40 in this issue), and authors of KnowledgePro, KnowledgeMaker, and MicroExpert, the distinction is no longer relevant to their work—they do most of their expert system work in an extended version of Pascal. Their article demonstrates that real world problems often require more flexibility from the language (and by extension the programmer) than you'd expect from the standard AI discussion. After talking it over for hours, Bev said, they'd had to admit that whatever you do, the program will eventually be translated into binary, and they've decided they no longer "believe" in AI languages, per se.

While on the subject of unbelievable things, you'll probably want to check out Edward Yourdon's newsletter, *American Programmer*. The premier issue just arrived and describes the publication as "dedicated to casting a caustic eye on the American software scene." It certainly does that! While Yourdon's perspective is a little more MIS oriented than many of us prefer, the newsletter is filled with perceptive commentary and intelligent discussion of such topics as the economics of the DP industry, and the issues of programmer productivity and costs.

One of the more startling forecasts in the premier issue is what Yourdon admits a rash prediction, the thesis that "the American programmer is about to go the way of the dinosaur and the dodo bird." Yourdon is talking here about programmers in the corporate environment, and he has a substantial body of facts and figures to buttress his thesis that foreign competition is about to do for programmers what it did for the American auto worker a few years back.

Obviously, at a charter subscription rate of (only) $295, this newsletter is not for the casual programmer. But a subscription for the company library would be a sound investment. It's good to be occasionally reminded that there are larger issues than the product announcements, advertisements, and reviews that many industry publications dwell on.

Tyler Sperry
editor

ARCHIVES

Ten Years ago in *DDJ*

"A company which claims it is ready to manufacture and sell truly intelligent household androids, or robots, within the next two years—to the tune of $4,000 apiece—has the nation's artificial intelligence experts up in arms. The experts unanimously state that the claims are fraudulent. The company, Quasar Industries located in Rutherford, NJ, is currently touring shopping centers around the country with a model of an android which they claim comprehends 4,000 words of vocabulary and can vacuum, wash dishes, and teach the kids French. Crowds in the sundry department stores are dazzled, according to numerous press reports. This robot is a fake, according to scientists from Carnegie-Mellon University in Pittsburgh, PA. The scientists recently undertook a first-hand investigation of the matter and found the robot to be a "radio-controlled puppet." Stanford University News Release, "Et Tu Klatu?"—*"Letters,"* DDJ, *February 1978.*

Just another arty fact

"After all, the strange fact about AI has always been that it's easier to simulate an expert than to simulate the general common sense of a five-year-old."—*Michael Doherty,* DDJ, *June 1984.*

It's always so code in here!

"*DDJ* is important as a journal because of the code. It was started to publish code and it has always published code. It publishes more code than any other magazine. And its still my belief that people [programmers] learn from reading other people's code.

What they learn, of course, is how to program well: they pick up tricks, insights, algorithms, all of which are particularly well expressed when presented in a form in which they will ultimately be realized: as code. We hope to go right on publishing useful and educational code, including both programs significant in themselves and bits of code that demonstrate some exemplary algorithm or insight."—*Michael Swaine,* DDJ, *February 1985.*

DR. DOBB'S JOURNAL of
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte

New! Introducing Turbo C 1.5— the best optimizing compiler gets even better!

The professional optimizing compiler for less than $100

Turbo C® is a technically superior production-quality compiler. (Borland's equation solver, Eureka™, is written in Turbo C.) And our Turbo C 1.5 offers a new library of the highest presentation-quality graphics in the industry—the kind you'll see in Quattro,™ our new professional spreadsheet.

And spectacular graphics are just part of the brand-new features. Turbo C 1.5 enhancements also include:

- A professional-quality graphics library of over 70 functions
- A librarian that allows you to build your own object module libraries
- Context-sensitive help for the language and the library routines

Actual photograph of Turbo C graphics displayed on IBM 8514 screen.*

- Text/video functions, including windows
- 43- and 50-line mode support
- VGA, CGA, EGA, Hercules, and IBM 8514 support
- File search utility (GREP)
- Sample graphics applications
- More than 100 new functions

For professional-quality C at an affordable price, no one else comes close to Turbo C. Because no one can deliver technical superiority like Borland.

60-Day Money-back Guarantee**

For the dealer nearest you or to order, call

(800) 543-7543

BORLAND INTERNATIONAL

BORLAND

Minimum system requirements: For the IBM PS/2™ and the IBM® and Compaq® families of personal computers and all 100% compatibles. PC-DOS (MS-DOS®) 2.0 or later. 384K.

*Artwork metafile courtesy of Genigraphics® Corporation

**Customer satisfaction is our main concern; if within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department, and we will arrange a refund.

BI 1165B

It's easy to upgrade to Turbo C 1.5!

Just complete this coupon and mail it with payment before June 30, 1988. Or, call us at (800) 543-7543 and be ready to give our operators your name, credit card number, and the serial number on your Turbo C master disk.

Turbo C 1.5 Upgrade Price ***$ 33.50***

CA and MA residents add sales tax _____

Shipping and handling
In US $5.00 (Outside US add $10) _____

Total amount enclosed $ _____

Must include your Turbo C serial # _____

Return this coupon and the Turbo C RTL source code registration form from your Turbo C manual along with your payment by March 31, 1988 and receive your Turbo C 1.5 upgrade for free! (No phone orders please.)

Turbo C 1.5 Runtime Library Source Code ***$ 150.00***

CA & MA residents add sales tax _____

Price includes shipping to all US cities.
(Outside US add $10) _____

Total amount enclosed $ _____

Please specify diskette size ☐ 5¼" ☐ 3½"

Method of Payment: ☐ VISA ☐ MC ☐ Check ☐ Bank Draft

Credit card expiration date: _____ / _____

Card # |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Name _____

Ship Address _____

City _____ State _____

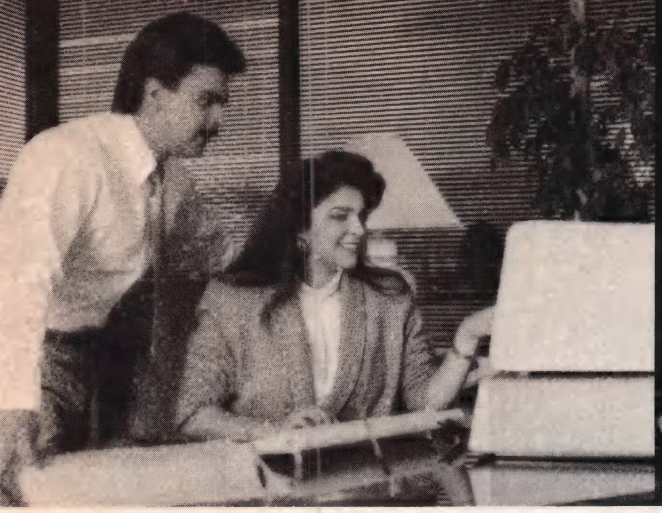
Zip _____ Phone (___) _____

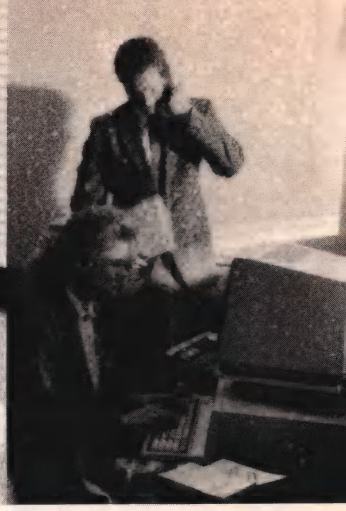
Mail coupon to: Turbo C 1.5 Upgrade Dept., Borland International
4585 Scotts Valley Drive, Scotts Valley, CA 95066

This offer is limited to one upgrade per valid product serial number. Not good with any other offer from Borland. Outside US make payments by bank draft payable in US dollars drawn on a US bank. CODs and purchase orders will not be accepted by Borland.

DDJ 4/88

CIRCLE NO. 106 ON READER SERVICE CARD





One Language For V

BB^x Specifications:

Ease-of-use—BB^x is the fastest, most powerful development tool available for business oriented program creation. Programmers can write code in minutes.

Execution time—BB^x's partially compiled format provides enhanced execution speed.

Easy Maintenance—BB^x is an interactive programming language, with a trace facility and a full screen editor which makes program maintenance a snap.

Portability—BB^x runs under UNIX and other operating systems without recompilation.

Compatibility—BB^x is an enhancement of the Business Basic language, an industry standard, giving you access to thousands of applications.

Supportability—Program maintenance utilities and complete documentation save considerable time and money. It lets you build and support applications easily.

Utilities—A complete set of BB^x utilities are provided for program and file management.

Conversion—A complete set of conversion utilities are provided with every BB^x package.

Features

Math Functions
- 14 place precision and computational accuracy
- Floating point conversion
- Task specified rounding precision
- Binary to decimal conversion
- Long function names
- Dynamic array's

String Functions
- Numeric to string conversion
- String manipulation
- No string length restriction

I/O Functions
- Windowing
- I/O mnemonics
- Device independent verbs
- X,Y cursor addressing
- Masking
- Soft key loads
- No record length restrictions
- BB^x file limit size is the size of the media which they reside

File Structures
- INDEX
- KEYED
- MKEYED
- SERIAL
- SORT
- PROGRAM
- STRING

System Structure
- Multi-tasking - which provides record and file level locking
- Program overlay
- Public programming which provides:
  - Local variables
  - Dynamically called sub-programs
  - Argument passing
  - Automatic public program drop from memory at exit
  - Public program in memory lock option

Language Structure
- Interactive program development
- Online syntax checking
- Compound statements
- User defined functions
- Unlimited nesting
- Remote I/O lists
- Program self modification
- Case insensitive console mode
- Various debugging tools

BB^x Utility Set
- File Browse
- Create Data Bundle
- Calculator
- Clear Workspace
- Program Compare
- Copy File
- Define/Redefine File
- Directory Listing
- Erase File
- Generate Filelist
- Program List/Cross Reference
- Move File
- Program Renumbered
- Rename File
- File Resizer
- Execute O/S Shell Command
- Search and Replace Program
- Color & FUNC Key Setup
- Time/Date Examine/Set
- Utility Menu
- Visual Utility Interface
- BXSND/BXRCV conversion utilities

Its portability crosses all operating environments, and now its performance is crossing all oceans.

Around the world, the industry's best and brightest programmers are discovering the astonishing power that BB^x brings to Business BASIC. Write your program once, and have complete movement to MS/PC-DOS, OS/2, XENIX-UNIX and VMS.

This year, over 50,000 copies of BB^x are performing throughout the United States, Canada, Europe, Asia and South America.

Commitment to innovation, development within industry standards and technological leadership have grown BB^x around the globe.

In 1988, aggressive marketing and uncompromising customer support will continue to compliment our success, and expand the BB^x standard among many of the world's most respected companies.

Get in touch with one of our world distributors, and feel the pulse of the power of BB^x!

Die Portabilität schlägt sämtliche, bisher bekannte und unbekannte, EDV-Emgebungen. Die Leistung überzeugt inzwischen die gesamte EDV-Industrie.

Weltweit entdecken die besten Software-Entwickler die erstaunliche Leistung von BB^x, mit der Business BASIC bereichert wird. Die Anwendungen werden nur einmal entwickelt und laufen ohne Änderungen oder Anpassungen auf MS/PC-DOS, OS/2, XENIX-UNIX, AIX, IX370 oder VMS.

Mehr als 50.000 BB^x-Lizenzen stellen die Leistung in den USA, Canada, Europa, Asien und Süd-Amerika unter Beweis.

Zur Innovation nach Industrie-Standard Spezifikationen verpflichtet, und mit dem Ziel nach technologischer Führung, wächst BB^x um die Welt.

Mit aggressivem Marketing ohne Kompromisse im Bereich Kundenservice, wird der Erfolg von BB^x in 1988 fortgesetzt. Es steht auf sämtlichen Systemen namhafter Computerhersteller zur Verfügung und stellt seine Akzeptanz bei den anspruchvollsten Anwendern unter Beweis.

Kontaktieren Sie unsere Vertretungen in aller Welt. Entdecken Sie die Schlagkraft von BB^x!

BB^x PROGRESSION/2 is available for Intel Based Computers, Altos, Arete, AT&T, PCS Cadmus, Computer Consoles, Convergent Technologies, Counterpoint/MultiTech Cubix, Data General, Digital Equipment, Fortune, Honeywell, Hewlett Packard, ICL, Motorola, Nixdorf, Prime, Pyramid, Rexon, Sanyo, Sequent, Siemens, Texas Instruments, Unisys, and the IBM family of products. BASIS is continually adding new systems.

orld Class Business.

Portable, il franchit tous les cadres d'opération, et sa performance traverse, maintenant, tous les océans.

Dans le monde, les meilleurs et les plus brillants programmeurs de l'industrie découvrent l'étonnante puissance que BB^x amène au BASIC des affaires. Ecrivez votre programme une seule fois, et accédez totalement à MS/PC-DOS, à OS/2, à XENIX-UNIX et à VMS.

Cette année, plus de 50 000 copies de BB^x fonctionnent aux Etats-Unis, au Canada, en Europe, en Asie et en Amérique du Sud.

Un esprit constant d'innovation, un développement conforme aux normes de l'industrie, et une position de leader dans le domaine technologique, tels sont les atouts qui ont contribué à la croissance de BB^x dans le monde entier.

En 1988, un marketing dynamique et un appui inconditionnel à notre clientèle continueront à couronner notre réussite, et à étendre le standard BB^x à de nombreuses sociétés parmi les plus respectées au monde.

Contactez l'un de nos distributeurs mondiaux et découvrez la puissance de BB^x!

Su portabilidad traspasa todos los medios de operación y ahora su funcionamiento esta cruzando todos los océanos.

Los mejores y más brillantes programadores del mundo, están descubriendo la asombrosa potencia que BB^x ofrece al negocio B[illegible]SIC. Escriba su programa una v[illegible]: y tenga movimiento completo a MS/PC-DOS, OS/2, XENIX-UNIX y VMS.

Este año, más de 50,000 copias de BB^x estan funcionando en Estados Unidos, Canadá, Europa, Asia y América del Sur.

Empeño de inovación, desarrollo en los estandards de la industria y superioridad tecnológica han hecho crecer a BB^x en todo el mundo.

En 1988, mercadotecnia agresiva y apoyo constante a nuestros clientes seguirán complementando el exito y desarrollo de BB^x entre las compañías más respetadas del mundo.

Comuniquese con uno de nuestros distribuidores mundiales y sienta la potencia de BB^x!

World Distributors:

Edias Hans Kirchhoff & Co. KG
Pfingstbornstr 25, 6200 Wiesbaden
TEL: (06122) 2016
FAX: (06122) 16505
TLX: 415-2563 edia d
West Germany, The Netherlands, Austria, Switzerland Denmark, Luxembourg, Belgium, England, Italy

Multisys
Torgeir Vraas Plass 5A
3044 Drammen Norway
TEL: (03) 83.86.05
FAX: (03) 89.02.53
Norway, Sweden, Finland, Greenland, Iceland

J.P. Brown and Associates
780 Gordon Baker Road
Willowdale, Ontario M2H 3B4
TEL: (416) 494-0472
Canada

PI Informatique
8, rue Benjamin Constant
75019 Paris, France
TEL: (1) 40.05.10.65
TLX: 214583
FAX: (1) 40.05.99.63
France, Spain, Portugal

Softech Pty. Limited
10 Eileen Road, Blairgowrie
Randburg 2194, South Africa
TEL: (011) 787-8839
South Africa

Risegold Pty. Ltd.
678 Paramatta Road
Croydon, N.S.W. 2132
Australia
TEL: (02) 799-6622
FAX: 21-3-40244068

Risegold Pty. Ltd.
86 Havelock Street
West Perth
Western Australia 6005
TEL: (09) 481-0607
FAX: 61-9-481-3162
Australia

Tempo Computadoras
Au. Americas #670
Guadalajara 44680
Jalisco, Mexico
TELS: 30-28-45
30-28-46
30-28-86

Infotel, S.A., DE, C.V.
Concepcion Beistegui 1959
Desp. 5
Col. Narvarte
C.P. 03020
Mexico, D.F.
TELS: 590-31-63
696-06-91
696-13-22
590-68-68
Mexico

In the United States:
BASIS Incorporated
P.O. Box 20400
Albuquerque, New Mexico 87154
TEL: (505) 821-4407
FAX: (505) 821-1625

BASIS™
INCORPORATED

10% DISCOUNT!
ANY ORDER BY A NEW CUSTOMER, ACCOMPANIED BY THIS ADVERTISEMENT, QUALIFIES FOR A 10% DISCOUNT!
Call TOLL FREE directly to our Order Department.
1-800-423-1394
DD

CIRCLE NO. 107 ON READER SERVICE CARD

LETTERS

Correcting our Pointers

Dear *DDJ*,

I could not bear to let Clyde Schechter's letter, Optimum Performance?, in the January 1988 issue go unanswered. It will leave young C programmers very confused about both multidimensional arrays and pointers.

Apparently Mr. Schechter has a two-dimensional array confused with an array of pointers. The definition:

```
int ary[10][10]
```

reserves storage for 100 integers, and the storage is quite contiguous. If the storage is not contiguous, then the concepts of pointer arithmetic and scaling in the C language have been built on faulty ground. The variable *ary* is an "array of 10 pointers, each containing 10 integers." The element *ary[0][10]* is the same element as *ary[1][0]*. No pointer variables have been created. The expressions *ary*, *ary[0]*, and so on are all pointer constants. As pointer constants their values cannot be changed, meaning that they cannot appear on the left of an assignment expression. Rows cannot be swapped by swapping pointers, as Mr. Schechter implies. The distinction between pointer constants and pointer variables is fundamental to fully understanding the C language.

I think the definition he describes is:

```
int *ptrary[10]
```

ptrary is "an array of 10 pointers to integers." The storage for the pointers has been reserved, but the pointers have not been initialized. The storage to which they point is not yet present. Dynamic memory allocation can be used to obtain different amounts of storage for 10 different arrays, and each pointer can be set to reference one of the arrays. Now the pointers (array elements) can be exchanged.

Your readers should understand the difference between these next two expressions. *ary[1][2]* references element (1 * 10) + 2, element number 12, (the 13th element counting from zero), of array *ary*. *ptrary[1][2]* references element number 2 (the third element) of the array pointed to by element number 1 (the second element) of array *ptrary*. The scale of each row of *ary* is 10, while the scale of the rows of *ptrary* is unknown.

In support of Mr. Schechter, this topic is one of the most confusing aspects of the language. In *C by Design*, the college level textbook which I am co-authoring, we decided to devote one full chapter to discussing arrays, and another to discussing pointers. It required that much space to explain these subjects. I hope that this letter provided an adequate overview, and thanks for letting me express this opinion.

George Defenbaugh Jr.
Tulsa, OK

Your points are well taken, and you were not alone in your concern. We'd planned to have a reply from the author, Richard Relph, but last minute changes in the January issue prevented that. —Ed.

Poor Richard's Revenge

Dear *DDJ*,

One of the other editors here at *BYTE* pointed out Tyler Sperry's article in your January 1988 issue, "386 v. 030: The Crowded Fast Lane." I was inexplicable drawn to that article—maybe you can guess why.

In his article, Tyler summarized the conclusions I came to in my September 1987 *BYTE* article (in which I benchmarked a number of 80 x 86 and 68xxx machines). To quote his summary: "If you have some experience with benchmarks (or if you read *BYTE* regularly), you can anticipate what he [meaning: me] found: the 80386 outperformed the 68020 in the majority of tests."

I'm not sure how Tyler deduced that, but he's flat wrong. My article said: "Overall, it appears that—and I know I'll catch a lot of flak for this—the 80386 machines outperform the 68020 machines. Of course, the reason for this could well go beyond the *possibility* [my emphasis] that one processor is simply faster than the other." Nowhere have I made the claim that the 80386 processor performs better than the 68020. I'm surprised that Tyler interpreted "the 80386 machines..." as "... the 80386 processor..." My article was an attempt to reveal aspects of performance of systems based on processors, not an attempt to isolate processor performance.

Further down, Tyler states (in reference to the benchmark tests I ran): "...these tests were performed with the intent to test mathematical performance. The only nonmathematical tests were the infamous Sieve and a quicksort routine."

I've got to throw a flag on that one as well; check out two of the other tests I ran: Dhrystone and Fibonacci. Dhrystone is not a purely mathematical benchmark. It contains a mix of operations that involve indirect addressing, comparisons, array operations, and string manipulations. Nor would I consider Fibonacci's only goal to test math operations; it's main objective is to test the instructions involved in recursion.

All in all, Tyler's article was right on. I couldn't agree more with many of his "lessons." I've always admired the work *DDJ* has done, you people put out one of the finest computer magazines around. Keep it coming, and the other *BYTE* editors and I will keep reading it. (Just ask Tyler to read *BYTE* more carefully next time.)

Richard Grehan
BYTE Magazine
Peterborough, NH

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

Call 1-800-45-VEDIT for FREE Fully Functional Demo Disk

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS easily satisfies the most demanding computer professional.

Try a Dazzling Demo Yourself.

The free demo disk is fully functional—you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial—you experiment in one window while another gives instructions.

The powerful "macro" programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a "macro"—it shows that no other editor's "macro" language even comes close. And VEDIT PLUS is only 40K in size.

Go ahead. Call for your free demo today, You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Only VEDIT PLUS is this Flexible.

The installation lets you pick from closely emulating the keyboard layout of Word Perfect, WordStar and others. Or you can easily create your own layout and even your own editing functions. Supports any screen size—you pick screen colors and attributes.

Supports the IBM PC, XT, AT and PS/2. Also supports MultiLink, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, FlexOS (protected mode), CP/M-86 and CP/M. (Yes, we support windows on most CRT terminals, including CRTs connected to an IBM PC.) Order direct or from your dealer. $185.

Special: VEDIT (single file, no windows) for CP/M—$49.

- **Fully Network Compatible**
- **Call for XENIX-286 version**
- **30 Day Money-back guarantee**

Compare Features and Speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	Unlimited
Multiple file editing	20+	2	No	20+
Windows	20+	2	No	20+
Macro execution window	No	No	No	Yes
Pop-up menus	No	No	No	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Automatic processing of Compiler errors	Yes	No	No	Yes
"Cut and paste" buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
Convert to/from WordStar	No	No	No	Yes
On-line calculator	No	No	No	Yes
Configurable Keyboard	Hard	No	Hard	Easy
43 line EGA support	Yes	No	No	Yes
Manual size/index	250/No	42/no	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec

*Also available for TI Professional, Tandy 2000, DEC Rainbow, Wyse WY700 and others.
*Demo disk is fully functional, but does not readily write large files.

CIRCLE NO. 108 ON READER SERVICE CARD

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, TELEX 701821

(continued from page 12)

Author Tyler Sperry replies:
Regarding your first point, I must apologize. As you point out, even though your article compared 386 and 020 systems you clearly warned your readers that the results could not be extended categorically. (Although it could be argued this is exactly what most of your readers wanted.) As the later paragraphs of my piece made clear, both of us agree on this point. Apparently, my left brain took a vacation during the proofing and missed the gaff.

As to the benchmarks, I must confess that characterizing the Dhrystone benchmark as "mathematical" was a mistake. The single aspect of the Dhrystone is the *strcmp* function—it can affect compiler benchmark performance by as much 30 percent—and comparing strings isn't what most of us consider "mathematical." My mistake with the Fibonacci benchmark was a sin of omission; I consider the Fibonacci to be virtually worthless as anything other than a quick-and-dirty test for the way a compiler handles procedure calls, and perhaps I should have spelled this out.

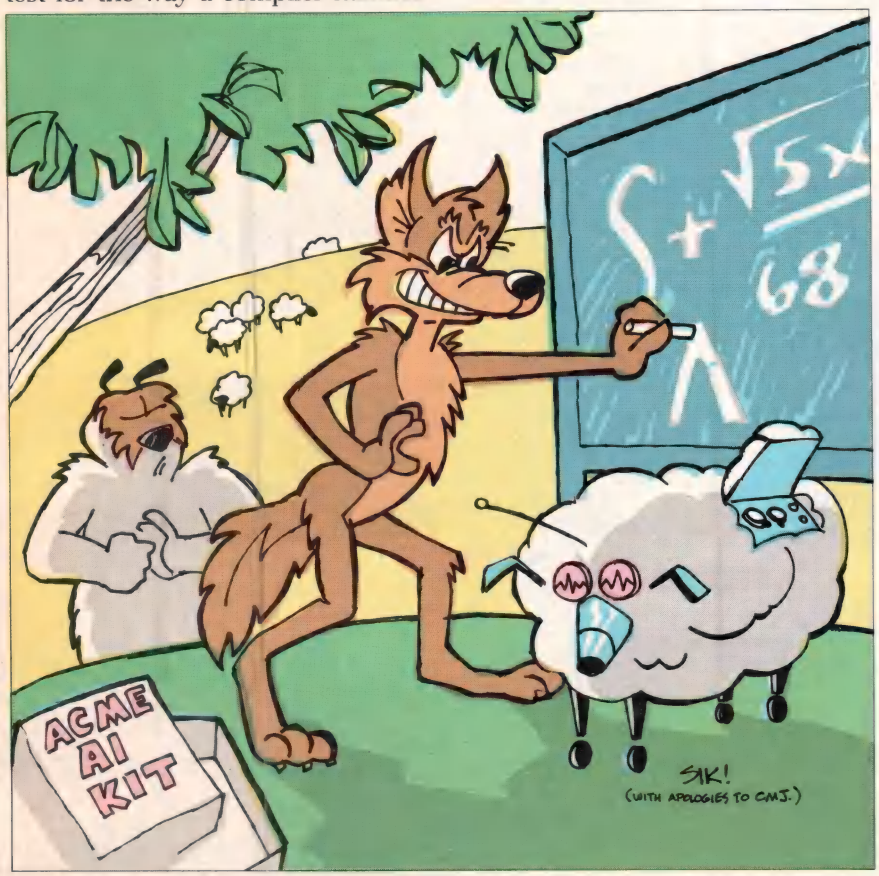
Despite these minor points, I think we agree on the broad issues. It's a pity we have to rely on benchmarks for comparisons of systems—or CPUs.

Dynamic Linking Revisited

Dear *DDJ*,
Congratulations on giving OS/2 some decent coverage in the December issue of *DDJ*.

However, Dave Cortesi's article ("Dynamic Linking in OS/2,") did contain one little flaw. He asserts that DLL s cannot be written in high level languages because of the requirement that SS = DS. In fact, Microsoft C has a switch especially to tell the compiler not to assume DS = SS, and Microsoft uses C to write many of the DLLs that are supplied with OS/2 and Windows.

Ray Duncan
Ex-member of the
Happy *DDJ* Family
Marina del Rey, CA



Early attempts at lambda calculus.

Dear *DDJ*,
I appreciate your article describing the dynamic linking mechanism in OS/2. One minor correction, though. The current C compiler for OS/2 (at least the IBM and the Microsoft offerings) can indeed generate code for dynamic libraries. One must use the *–Au* and *–Gs* options. The *–Au* (usually -Alfu for large model) generates code for SS!=DS. Admittedly not all library modules are available (in particular the I/O library), but one can write shared code in C. With the *–Au* option the compiler generates a save and load for DS for each procedure. *–Gs* suppresses stack checking.

Also, a quick comment on your enthusiasm about the 80386. Yes, it is indeed a wonderful processor. But both OS/2 and the VM systems (Windows/386, Desqview and others) fail to fully exploit the chip. While the VM systems' value comes from compatibility with existing applications (not a minor point), they do not otherwise allow applications to exploit the processor. To fully exploit the native mode (also called protected mode) the path that OS/2 provides has the longterm advantage. Obviously, a merger of the two approaches is needed.

Robert Frankston
(via CompuServe)

Author Dave Cortesi responds:
Duncan and Frankston are correct; both Microsoft C 5.0 and IBM C/2 support the *–Au* option. That generates code at every function's entry to save DS and load it with the selector for the data segment generated for the module, and code to restore the caller's DS on exit. As Frankston notes, the *–Gs* option eliminates the check on stack depth, and that eliminates the commonest source of link-time references to unwanted library procedures.

These options are, however, peculiar to the particular compilers. The IBM Pascal/2 and comparable Microsoft Pascal compilers do not have them, and other C compilers might or might not. And while the features help, they don't fill all the potholes on the road to dynamic linking.

(continued on page 142)

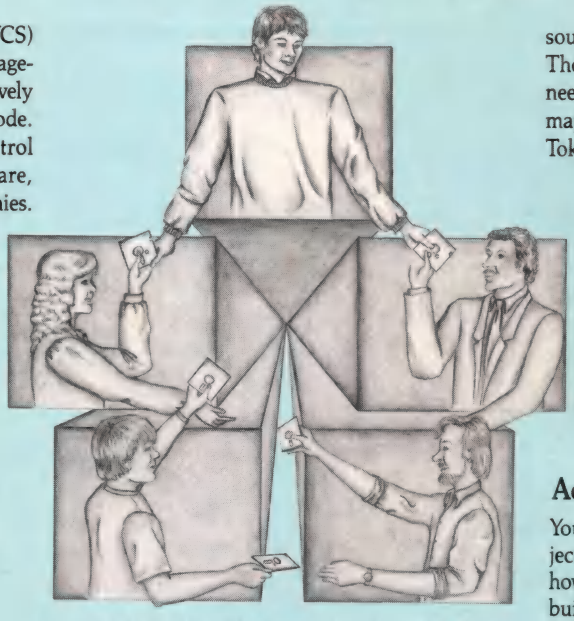
PVCS

The Number One Source Code Control System.

The POLYTRON Version Control System (PVCS) simplifies and automates Configuration Management so programmers and managers can effectively control the revisions and versions of source code. PVCS is the most widely used change control product and is used by the leading software, aerospace, manufacturing and service companies.

"In terms of features, PVCS provides everything necessary to a large multi-programmer project — more than any other package reviewed. No restrictions are placed in the development environment and all aspects of operation can be customized for specific project needs."

PC Tech Journal
September 1987



Unmatched Flexibility

- Storage & Retrieval of Multiple Revisions of Source Code
- Maintenance of a Complete History of Changes
- Control of Separate Lines of Development (Branching)
- Resolution of Access Conflicts
- Optional Merging of Simultaneous Changes
- Release and Configuration Control
- Project Activity Reports
- Management Reports
- Command or Menu Interface

Project Control

PVCS maintains individual archieves of all project components in your system — source code modules, data files, documentation and even object code libraries. These "source documents" can be written in any language or multiple languages.

Fast Retrieval of Revisions

PVCS uses "reverse delta storage" which saves disk space and speeds retrieval of versions of any file in the project database. A delta is the set of differences between any revision and the previous revision. PVCS can rapidly recreate complete versions of any file whether it is the most recent revision of a module or the original version of the entire project. Differences are automatically detected and stored.

A Practical Necessity for LANs

While important for single-programmer projects, PVCS is absolutely essential for multiple-programmer projects and LAN-based development efforts. In a LAN environment, source code files are simply too easy to change. Because any change to any file can have major ramifications, coordinating and keeping a record of changes is critical. Project leaders can determine, on a module-by-module basis, which programmers can access or modify source files, libraries, object code and other files. The levels of security can be tailored to meet the needs of nearly every project. PVCS works on all major LANs including 3Com, Novell and the IBM Token Ring Network.

"PVCS has helped us maintain nearly 90 programs and utilities. Without it we would not have the quality of our upcoming release of NetWare."

Jonathan Richey
Manager, NetWare Utilities
Novell

Adopt PVCS on Your Existing Projects

You can obtain the benefits for your current project without disrupting development, regardless of how long your project has been under way. You can build PVCS archives from revisions stored in your present files or simply adopt PVCS from the current date.

PolyMake Reads PVCS Logfile Format

PolyMake, the leading Make utility, understands the structure of PVCS logfiles and is able to correctly determine the date and time of any revision. This prevents unnecessary operations that occur when the date and time of the complete project archive itself is used as with other make utilities.

Once you standardize on PVCS, the archives used to track and monitor changes are interchangeable between any PVCS product.

Personal PVCS — Offers most of the power and flexibility of Corporate PVCS, but excludes the features necessary for multiple-programmer projects.

Corporate PVCS — Offers additional features to maintain source code of very large and complex projects that may involve multiple programmers. Includes multi-level branching to effectively maintain code when programs evolve on multiple paths.

Network PVCS — Extends Corporate PVCS for use on Networks. File locking and security levels can be tailored for each project.

PVCS for VAX systems — Requires VMS. Uses the same interface and archive format as MS-DOS version. Supports branching and offers file locking and other security features for multiple-programmer projects.

	MS-DOS*	VMS		
	PC/XT/AT	Micro VAX II	VAX 7xx	VAX 8xxx
Personal PVCS	$149			
Corporate PVCS	$395			
Network PVCS	$995**	$4,950	$9,500	$10,500+
PolyMake	$149			
Network PolyMake	$447**	$1,250	$2,375	$2,500+

**5 Station LAN License. Call for pricing on larger Networks.

TO ORDER:
1-800-547-4000
Dept. DDJ

Oregon & Outside USA call (503) 645-1150.
Send Checks, P.O.s to: POLYTRON Corporation, 1700 NW 167th Place, Beaverton, OR 97006

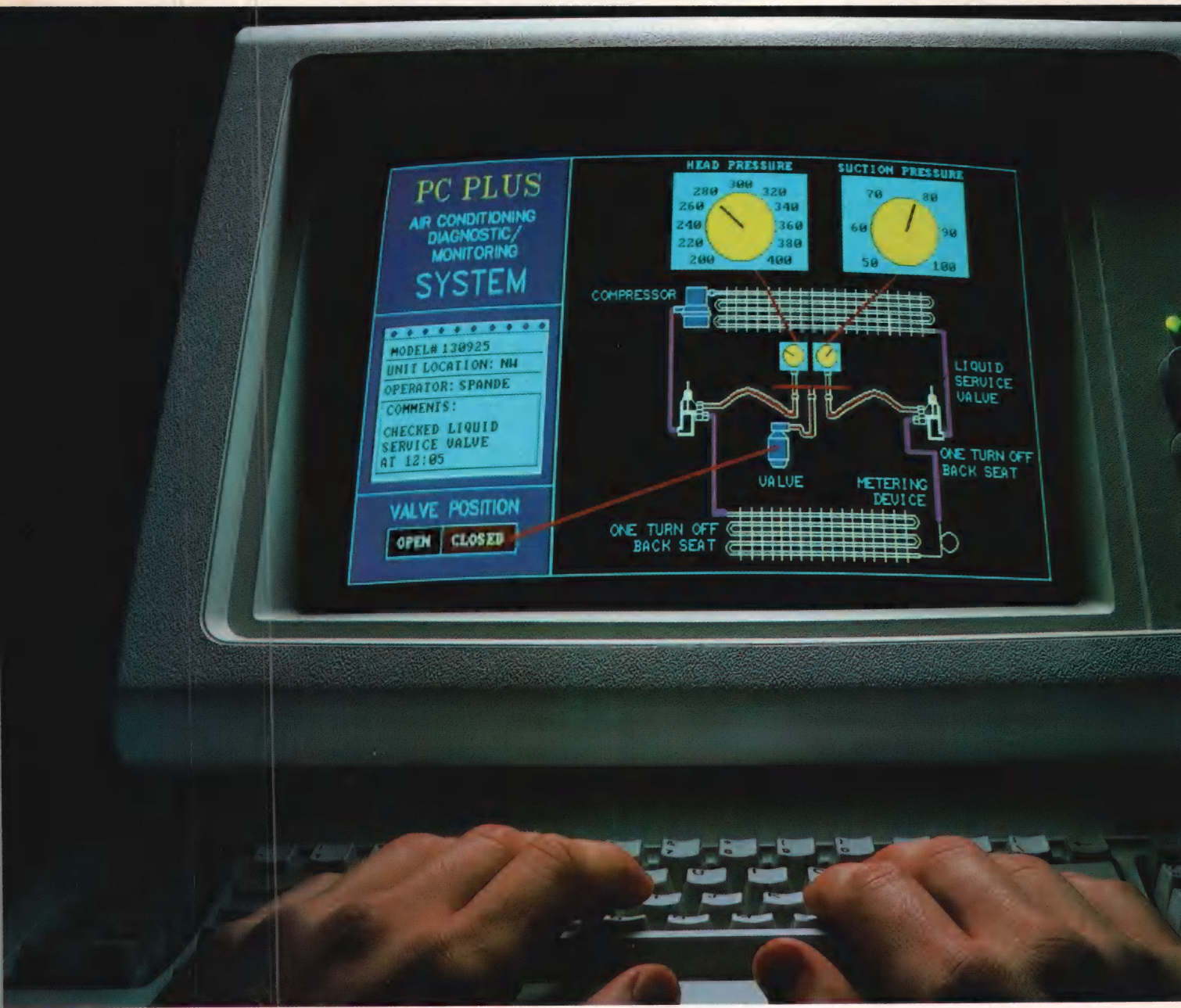
®

POLYTRON

High Quality Software Since 1982

CIRCLE NO. 109 ON READER SERVICE CARD

Texas Instruments has system developers need.



"Personal Consultant™ Plus…offers a very fine expert system development and delivery tool that already has a proven record with end-users."

— *Susan Shepard, AI Expert*

Personal Consultant Plus 3.0 Standard Features
- Frames, rules, meta rules and procedures
- Forward/backward chaining
- Confidence factors
- Regression testing and rule tracing
- End-user explanation facilities
- Graphics image capture and display
- Interfaces to dBase™, Lotus 1-2-3™, DOS files, .EXE or .COM programs, "C"
- Complete LISP development environment
- 2-megabyte expanded/extended memory support
- Mouse support
- Context sensitive help
- "Getting Started" tutorial-style manual

Personal Consultant Images
- Optional add-on package to PC Plus (3.0)
- Allows integration of "active images" into

what serious expert Power tools.

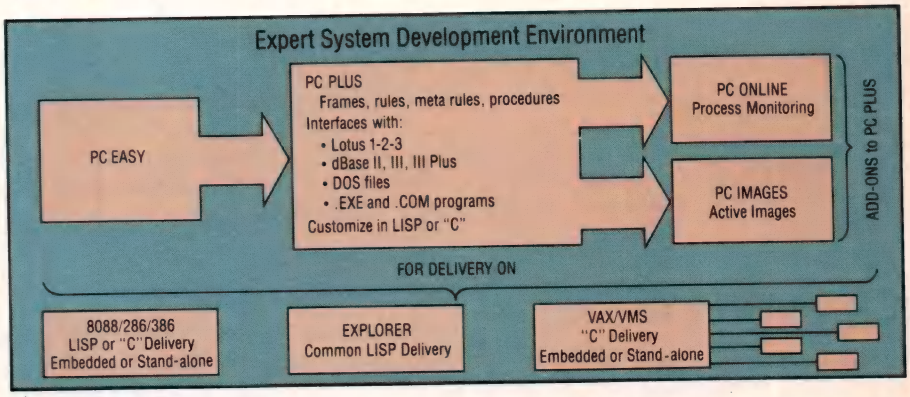
knowledge bases
- Interactive dials, gauges, forms and selection images
- Multiple images can be combined on same screen
- "Getting Started" tutorial-style manual

Personal Consultant Online
- Optional add-on package for PC Plus (3.0)
- ONLINE expert systems that interact directly with process data
- Multiple interfaces to data acquisition and analysis programs
- Knowledge base synchronization with process data
- Functions for historical and predicted trends
- Special user interface/reporting capabilities
- "Getting Started" tutorial-style manual

Among all the expert system development tools available for personal computers today, none deliver the power and flexibility of TI's Personal Consultant series.

Personal Consultant Easy is ideal for getting started, and is upwardly compatible with the higher functionality of PC Plus. For experienced developers, Personal Consultant Plus and its optional add-on enhancements, Online and Images, were designed to help solve a broader range of complex problems.

Application delivery as flexible as the tools themselves.
Delivery can be in LISP for flexibility, or "C"* for maximum speed and portability. Our "C" options support either stand-alone or "embedded" knowledge bases. Options are available for DOS-based PCs, TI's Explorer, and DEC's VAX™ line of multi-user minis running under VMS™.



Personal Consultant Plus. Full power for an affordable price.
At $2,950, PC Plus has proven to be one of the richest and most flexible problem-solving tools available for the development of complex knowledge-based systems. Designed to take advantage of today's more powerful 286/386 DOS-based computers, or TI's Explorer™ Symbolic Processing System, the new 3.0 version of PC Plus provides powerful standard features and a continuing growth path with the addition of either PC Images or PC Online, or both.

Personal Consultant Images. Picture an expert system with interactive graphics.
At $495, PC Images enables developers to create knowledge-based applications that incorporate complex graphical "active images." User-interactive dials, gauges, forms and selection images provide a more exciting visual data input and output style.

Personal Consultant Online. The expert system as part of the process.
At $995, PC Online allows the developer to design expert systems which interact directly with process data, as opposed to input from a human operator. Designed for intelligent process monitoring applications, this optional package helps deliver expertise that is "online all the time."

"Texas Instruments has done more than any other company to educate people about AI, to popularize it, and to make useful AI tools available at reasonable prices."
— Jim Seymour, PC Magazine.

Technical support, training courses and Knowledge Engineering Services are available for the Personal Consultant products. If you have a question about any of our expert system power tools, we have the answer.

Pick up the phone and gain a powerful advantage.
Call **1-800-527-3500** for technical overviews of our products and a PC Plus case histories brochure which details how our power tools are being put to work today.

36106

*Available 4Q 1987.

TEXAS INSTRUMENTS

Creating an Adventurous Language

by Jonathan Amsterdam

While browsing around my local computer system not long ago, I had the misfortune to stumble upon the source code for the original Crowther-Woods Adventure program. It was a gut-wrenching display of nonmodularity run rampant. Except for the text of the messages and the network of room interconnections, the entire game, from the properties of rooms and objects to the movements of the bear, troll, pirate and dwarves, was coded right into the program.

I'm sure I wasn't the first to be goaded by this horror into designing a special-purpose language for writing adventure games. But mine differs from others I have seen in its attempt to combine the best of three different programming styles into a single, unified, Adventure Authoring Language, AAL. In this article, I describe AAL (pronounced simply "Al"), emphasizing the roles of the programming styles that comprise it and their implementation challenges.

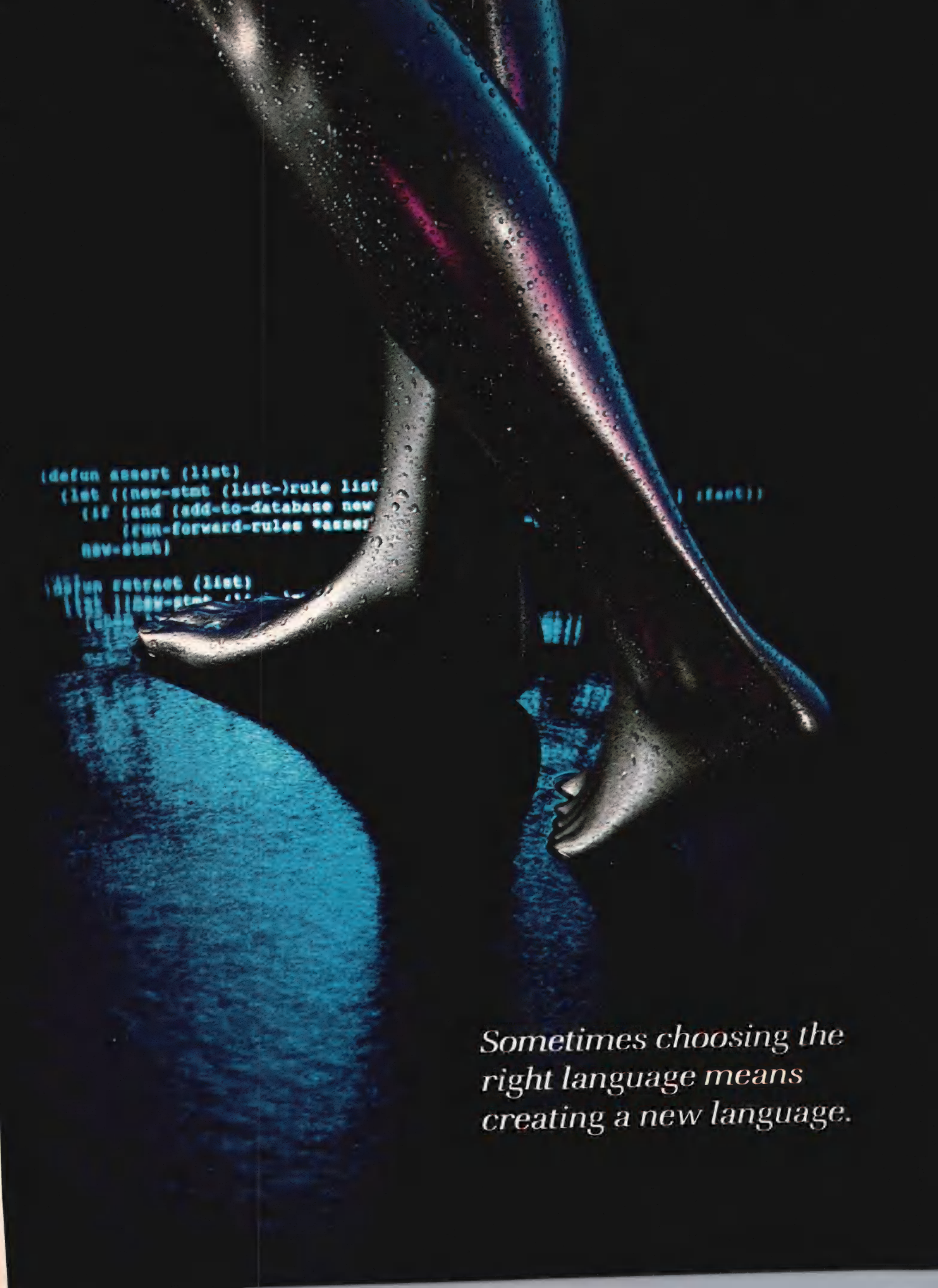
The programming styles used in AAL all arose from artificial intelligence (AI) research of the 1960s and 70s. Today, each style is typified by a single programming language. The core of AAL involves deductive retrieval from a database of facts and rules, as popularized by the logic programming language PROLOG. AAL borrows the idea of inheritance from object-oriented languages such as Smalltalk for describing the objects of adventure games. And the list-manipulation abilities, syntactic freedom, and general-purpose power of LISP make it the ideal language for implementing AAL.

First, I'll supply an overview of AAL, to give you an idea of how these aspects of the language are realized. Then I'll discuss the programming styles in slightly more detail. Finally, I'll consider some key points of implementation. Because of space limitations, I won't be able to describe all of AAL or its implementation.

Overview of AAL

AAL is designed for writing text adventure games in which players move from location to location and manipulate objects by typing brief commands. AAL maintains the state of a running game in a list of facts called the database. Portions of AAL programs can query the

Jonathan Amsterdam is a graduate student at MIT's artificial intelligence laboratory. He has articles published in several magazines, and may be reached at 617-253-7881.



Sometimes choosing the right language means creating a new language.

database to find out about the current state and can add (*assert*) and delete (*retract*) facts to change the state. Facts are represented by lists of symbols; the fact (*in keys house*) could mean that the keys are in the house.

Patterns are used to query the database. A pattern is like a fact but may contain variables, which in AAL begin with an asterisk. A pattern matches a fact if all the constants are identical; the result of a match is to bind the variables in the pattern to the corresponding constants in the fact. So, for example, the pattern (*in *x room*) matches (*in bear room*) with *x bound to bear but does not match (*in bear house*) because *room* and *house* are not equal.

AAL programs cause actions to happen in the adventure game by examining and modifying the database using rules. The rule:

```
((at lamp *x) (on lamp) -> (lit *x))
```

says that if the lamp is at a location *x, and the lamp is on, then *x is lit. Rules consist of zero or more antecedents (patterns to the left of the ->) and one or more actions or consequents. If all the antecedents match the database, then the actions are taken. Variables appearing in the actions have the same values that they were bound to when the antecedents were matched.

Rules can be combined into rule lists, which act like nested *if. . .then. . .else*s. The rule list:

```
(((carrying player rod) -> "The bird is frightened")
  ((not (carrying player cage)) -> "You can't carry it")
  (-> (take player bird)))
```

specifies what happens when the player tries to take the bird in the original Adventure. If the player is carrying the rod, the message "The bird is frightened" is printed and nothing else happens. If the player is not carrying the cage, the message "You can't carry it" is printed. If neither of these conditions hold, then the player takes the bird. (A rule with no antecedents is like an *else*: it always fires. It is also OK to omit the -> entirely in this case.) In general, a rule list can have any number of rules. Their left-hand sides are examined starting from the top, and the first rule to match is activated.

There are several different kinds of actions; you have already seen three. If a string occurs as an action, it is printed. If the first symbol in an action is the name of a built-in AAL procedure, such as *take*, then that procedure is executed (in the case of *take*, the procedure alters the database by asserting the statement [*carrying player bird*] and removing the statement that says in which location the bird is). If an action begins with the word *lisp*, it is given to the LISP interpreter for evaluation; this is an easy action to implement because AAL is written in and runs on top of LISP. If the action can't be classified as one of the above, it is assumed to be a statement to be asserted, as in the case of (*lit *x*).

AAL programs consist of descriptions of the locations, objects, and commands in the adventure game. Listing One, page 58, shows a short AAL program for a simple adventure with only two locations.

The *loc* form is used to describe locations. The first symbol after the word *loc* is the AAL identifier for the location—every entity in an AAL program must have a unique identifier. The next item in the list is a string giving the long description of the location; the identifier, slightly modified, serves as the short description (*the-first-room* is modified to "The First Room").

The remaining elements of the *loc* form are keyword lists—that is, lists beginning with special AAL keywords. A list beginning with the symbol *contains* specifies the items that the room contains at the start of the game. The *exits* list specifies the actions to take when the player attempts to leave the location. Each element of the *exits* list is itself a list, whose first element is the direction the player wants to go (for the first room, west and south are the only choices) and whose remaining elements provide the actions to take.

If an action is a symbol, it is assumed to be the identifier of another location and the player is transferred there without incident. If an action is a string, the string is displayed. So, if the player is in the first room and tries to go west, the list (*w the-second-room*) is examined and results in the player's being moved directly to the second room. If the player goes south from the first room, a message is printed and the player is left where he was. As the north exit from the second room demonstrates, rule lists can be used to implement more complex actions.

The third form in Listing One describes the *blow* command, which the player can enter to blow an object (such as the whistle—see later). The list (*blow *obj*) indicates the syntax of the command: the verb is followed by a single thing, the object of the blowing, which is bound to the global variable *obj*. The *requires* list specifies what conditions must hold for the command to be carried out; this one says that the player must be carrying the object. If not, a message is printed that varies depending on what the object is. This is accomplished using the syntax of Common LISP's *FORMAT* function, a flexible output system similar to but more powerful than C's *printf* function. Because AAL is implemented in and runs on top of LISP, it is very easy to adopt *FORMAT* directly into the language. The last line of the *blow* command description specifies the default action to take; in this case, it is to print a message.

The next form describes the *throw* command. The synonyms *hurl* and *chuck* can be used by the player, but internally only *throw* is used. *Throw*'s syntax is a little more complicated than *blow*'s: you say *throw <some object> at <something>*. The variable **instr* is set to the object thrown and the variable **obj* to the thing at which it is thrown. To throw something, the player must be carrying that thing, and the target must be *here* (at the player's location).

The fifth form in Listing One is an *obj* form describing an object—the game's monster. Each item in an *obj* form is either a feature or a keyword list. In this example the monster has one feature, *fixed*, indicating that it is fixed in place and so can't be taken by the player. Each object in an AAL program may have several features,

Some of the world's biggest problems are being solved with a touch of Smalltalk.

On the ground floor of high-tech environmental control.

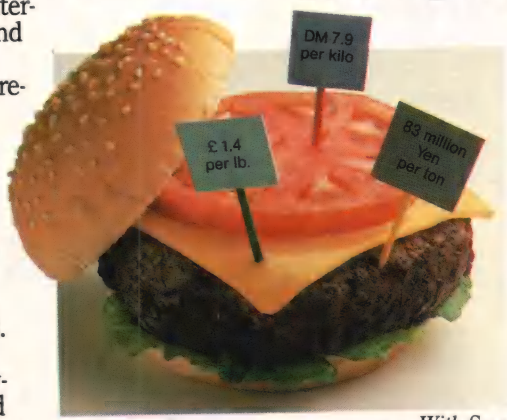
Climate, energy, fire and security are all critical aspects of environmental control in large office buildings. The challenge for Johnson Controls, a leader in this industry, is to provide a control system that is both technologically advanced and simple to operate. Using Smalltalk/V, Research Scientists Gene Korienek and Tom Wrensch have created a workspace environment that allows rapid prototyping and modeling of future systems. At Johnson Controls this system is used to explore relationships between cognitive models of building operators and corresponding iconic representations of building components. Each system can then be tested by simply clicking a mouse and viewing the results in sophisticated color graphics on a PC.

Abroad and involved in foreign affairs.

The French Ministry of Foreign Affairs is responsible for keeping track of every French citizen living abroad and every foreigner living in France. Each day, they process thousands of requests for documents or information, each one of which takes at least fifteen minutes. Arthur Andersen, the world's largest accounting firm, has developed a natural language processing application with Smalltalk/V that enables clerks without computer training to extract the necessary data much faster. Thanks to Smalltalk and system developers Bart Schutte and Pascal Wattiaux, what once took fifteen minutes now takes 30 seconds. Vive la Smalltalk!

The world is made of objects. So naturally, the world is turning to Object-Oriented Programming (OOPS). And the fastest, easiest OOPS language and environment is Smalltalk/V.

With OOPS you program by defining objects, their interrelationships and their behavior. Objects can represent both real-world entities —people, places, things —as well as useful abstractions such as stacks, sets and rectangles. Smalltalk/V provides everything you need to solve problems big and small, including a comprehensive tutorial to get you started.



Who needs Smalltalk?

Because Smalltalk models the way people really think, it is perfect for scientists, engineers and professionals who have to solve tough problems in a short amount of time. Perfect for programmers who are looking for a fast, efficient prototyping environment. And anyone who wants to quickly and easily learn OOPS.

Introducing Smalltalk/V286.

Our newest version of Smalltalk offers faster and more powerful OOPS capabilities. We've gone from 16 to 32-bit architecture. From 640K to 16 MB capacity for 25 times the memory. And designed it to run on the next generation OS/2 operating system as well as DOS.

Get Smalltalk for a small price.

Smalltalk/V sells for just $99.95. Smalltalk/V286 is $199.95. The following optional applications packs are available for $49.95 each: Communications; EGA/VGA Color; Goodies #1, Goodies #2, Carleton Tools and Goodies #3, Carleton Projects.

And everything comes with a 60-day, money-back guarantee.

So visit your nearest dealer. Or call toll-free, 800-922-8255 and order direct with MasterCard or Visa. Or write to Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045.

And let us help you put Smalltalk into action.

Teaching students to think economically.

With Smalltalk, even non-programmers can create exciting applications. Economics Professor Arnold Katz of the University of Pittsburgh developed Economics PC Discovery World, an intelligent tutoring system for beginning microeconomics students. Using a mouse to access windows and manipulate data, a student can call up a set of markets and commodities for an imaginary community. By changing the scenario, the student can not only study a variety of market behaviors, but also test the validity of his or her own reasoning. A process that provides a lot of food for thought.

Smalltalk/V

digitalk inc.

CIRCLE NO. 110 ON READER SERVICE CARD

Smalltalk/V requires DOS and 512K RAM on IBM PC/AT/PS or compatibles and a CGA, MCGA, EGA, VGA, Toshiba T3100, Hercules, or AT&T 6300 graphic controller. A Microsoft or compatible mouse is recommended. Smalltalk/V286 requires a 286 or 386, DOS or OS/2 and 1 MB of memory and one of the graphic controllers listed above. Not copy protected.

which describe various aspects of the object. For instance, a bottle might have the *container* feature, which allows the player to put things inside it; or a door, grate, or chain might have the *lockable* feature, which says that it can be locked and opened with a suitable instrument.

Features also act as predicates; giving the monster the *fixed* feature will result in the fact (*fixed monster*) being asserted at the beginning of the game. In the monster description, there is also one keyword list that describes what to do when the monster is the object of a *throw* command: the monster destroys what is thrown, and a message is displayed.

The next form defines a feature, *treasure*, which describes how to figure the score of an object (using the method of the original Adventure). Because many objects can share the same feature, features provide for great economy of coding. Here you see also that features can take arguments, allowing them to be adapted to different situations.

The next form describes the whistle, indicating that when it is blown, it emits a screech and kills the monster if it's present.

As you can see from the definitions of the *throw* and *blow* commands, the monster, and the whistle, the actions to take on a command are distributed throughout the program. Here's what happens when a command is entered: first, it's parsed according to the template supplied by the verb, and up to three variables are assigned: **command* to the command verb, **obj* to the object, and **instr* to the instrument. The last two are assigned as specified in the syntax template of the command.

Processing the command then begins, in two stages. First, the requirements are checked. The requirements specified with the command itself are checked first, then those on the object, and finally those on the instrument. In my example only the commands themselves have requirements, but it is common for special objects or instruments to place their own constraints on commands.

If all the requirements are satisfied, the command is executed. First, the object is checked to see if it has any actions; if so, they are carried out and processing stops. If the object has none, the instrument is tried, and if it has none, the actions on the command are done. In this way, objects implement their own actions for the most part, and actions specified with commands are the defaults.

This overview of AAL has hit the major features of the language, but unfortunately I have had to omit some features and many details. You will meet a few other aspects of AAL in the rest of the article.

AAL's Programming Styles

Having seen what AAL looks like, let's examine its roots. As I said, AAL combines aspects of three programming styles: logic programming, object-oriented programming, and LISP. Let's start with logic programming.

Logic Programming

In the late 1960s, it was noticed that mechanical theorem provers could be harnessed to answer questions about a database of information.[7] Further refinements led to several powerful AI programming languages, including PLANNER,[2] Conniver,[3] and AMORD.[4] PROLOG is a cleaned-up (but watered-down) successor of these languages. The basic idea behind deductive retrieval is simple: a query—typically a pattern containing variables—is compared against a database of facts, and those facts that match the query are returned.

You can add rules to the picture to make things more interesting (and complicated). There are two kinds of rules: backward rules, which are activated by queries, and forward rules, which are activated when new facts are added to the database. AAL has both these kinds of rules, though the earlier description didn't mention them.

Backward rules have a single consequent and one or more antecedents and are written in AAL with the consequent on the left, like so:

```
((in2 *x *y) <– (in *x *z) (in *z *y))
```

This rule says that something **x* is *in2* something else **y* if **x* is in some third thing **z* and **z* is in **y*. Backward rules are the rules of PROLOG; you could write the preceding rule in standard PROLOG syntax as:

```
in2(X, Y) :– in(X, Z), in(Z, Y).
```

Backward rules are stored in the database along with facts. When a query comes along that matches the consequent of a backward rule, processing continues with the rule's antecedents treated as subqueries. So, if your database consists of the facts:

```
(in water bottle)
(in bottle house)
(in food bag)
(in bag house)
```

and the preceding rule, then the query (*in2 water house*) would match the consequent of the rule, binding **x* to *water* and **y* to *house*; then the query (*in water *z*) is processed and matched against (*in water bottle*) with **z* bound to *bottle*; and finally, the second antecedent of the rule, which is (*in bottle house*) given the current variable bindings, is processed and matched against the identical fact in the database.

Note that the last two facts in the database provide a second answer to the query; if desired, this answer can be found via backtracking. Note also that with rules present, you have to extend the simple pattern-matching algorithm to deal with the case of matching two unbound variables against each other. In this case you simply bind the variables to each other; when either one becomes bound, the other is automatically bound to the same value. This generalized matching process is called unification.

So far, everything I have said is exactly as in PROLOG. Forward rules, however, do not exist in PROLOG (though

BREAKTHROUGH

COMMON LISP

FOR MICROCOMPUTERS

Develop and deliver professional AI applications on the Macintosh with Allegro CL ***for only $600.***

We're revolutionizing the economics of AI.

With Allegro Common LISP, you can develop your applications in a first-class environment and deliver them on the lowest-priced platform ever! For only $600, Allegro CL brings the complete Common Lisp standard and the best features of high-end LISP environments to the entire Apple Macintosh family of personal computers.

You Get a Full Common Lisp

Unlike other implementations of "Common LISP" for the Macintosh and IBM PC, Allegro CL is truly complete. No features have been left out. Allegro CL delivers lexical closures, complex numbers, adjustable arrays, and all other features of Common Lisp as specified in *Common LISP: the Language* by Guy Steele Jr.

With the Best Environment.

The Allegro CL programming environment is completely integrated with the Macintosh user interface to give you tools and features previously found only on high-end LISP machines. A programmable EMACS-style editor and other tools—such as a stepper, debugger, and a window based inspector—help you get your programs up and running quickly and easily.

Ask any of our customers, many of them major corporations and leading-edge companies in AI research. They'll tell you that Allegro CL for the Macintosh is the best programming environment this side of $60,000.

Allegro CL is Fast, Compact

Until now, the smallest Common LISP for a microcomputer was 3 megabytes of code. On workstations, Common LISPs routinely occupy between 5 and 8 megabytes. Allegro CL changes all that: it fits on a single 800K floppy and runs on Macintoshes with just 1 megabyte of RAM!

Our tightly written code coupled with our incremental compiler make Allegro blaze. Many developers report compile times on a Mac II that bettered LISP machines and left other micros far behind.

And Features Advance Interface Tools.

Within a matter of days, you can have a full working model of your program's user interface. Allegro CL provides a complete set of Macintosh interface components as pre-defined, self-managing objects. You also get full low-level access to the Macintosh Toolbox. Easier implementation means you can spend more time on design and refinement. Your product—and your customers—benefit.

You Also Get Our Commitment to Continued Development.

At Franz Inc., we're in the business of bringing you leading edge technologies on accessible hardware. To that end, our first additions to Allegro CL are already available.

The Allegro Flavors module gives you compatibility with Flavors release 6.1 code libraries. The Allegro Foreign Function Interface lets you call routines written in C, Pascal, or Assembly language.

It's Yours For Next to Nothing.

We are proud to offer Allegro Common LISP at an exceptional price. Call us today. Tomorrow, you could be testing the full capabilities of our revolutionary Common LISP environment.

For more information on how you can get started in Artificial Intelligence on the hottest micro in the market, contact us at:

Franz Inc., 1995 University Ave., Berkeley, CA 94704
(415) 548-3600. For orders and info,
call (800) 33 FRANZ (333-7260)

Coral Software Corp., P.O. Box 307, Cambridge, MA 02142,
(617) 547-2662

FRANZ INC.

Intelligent Software Tools

(continued from page 22)

they did in PLANNER and related languages). In AAL, a forward rule looks like this:

```
(when-asserted (in *x *y)
  -> (contains *y *x))
```

This rule says that if something **x* is in something else **y*, then **y* contains **x*. More precisely, whenever a fact of the form *(in *x *y)* is added to the database, this rule is activated and the actions on its right-hand side are taken. In this case, the only action is to assert another fact into the database. Backtracking never occurs with forward rules. Forward rules are similar to the production rules of languages such as OPS5.

Note that the rules in Listing One are strictly speaking neither forward nor backward because they are activated by commands, not queries or assertions. But because they behave more like forward rules, they have a similar syntax.

Object-Oriented Programming

Object-oriented programming, typified by languages such as Smalltalk and C++, consists of two major ideas: a program consists of a network of objects that communicate by message passing, and these objects obtain many of their attributes by inheritance.

The second of these ideas is an outgrowth of AI research into frame representation languages,[5,6] and it is the one AAL employs. Objects can have more than one feature, and features may themselves have features, allowing an interconnecting web of inheritance. As in standard object-oriented programming languages, inheritance facilitates the abstraction of common behaviors and properties.

LISP

LISP's enormous power derives from several sources. One source is LISP's ability to model a wide variety of abstractions using higher-order procedures, a feature best exemplified in the Scheme dialect[7] but also available to a large extent in Common LISP. I use this ability to implement *streams*, a data type crucial to AAL's implementation.

Other sources of power are LISP's great syntactic flexibility and list-processing power. Using these, it is possible to implement languages on top of LISP very easily, without the need for bulky and complex lexical analyzers and parsers. Instead, you let LISP's macros and *READ* function take care of that tedium and work at the much more convenient level of lists. And because LISP's interpreter is always present, even when AAL programs are running, you can easily allow AAL programs to escape into LISP, thereby in effect making AAL a superset of LISP.

Implementing AAL

There are numerous interesting aspects of AAL's implementation, but here I only have space to discuss a few of them. The most difficult and interesting part is the deductive retriever, or deducer as it's called in AAL, so I'll consider that first.

Listing Two, page 58, shows the entire code for the deducer. The interface to the rest of AAL consists of the four functions *assert*, *retract*, *deduce*, and *deduce-pattern*. *Assert* is responsible for adding a fact or rule to the database. It only adds something if it is not already present. If it does add a fact, assert checks the left-hand sides of the forward rules to see if any apply and executes the actions of all those that do apply.

Retract removes a fact or rule from the database if it is there. AAL also allows rules to trigger when facts are retracted, and *retract* handles this.

Deduce and *deduce-pattern* are far more complicated, so I'll look at them in great detail. *Deduce* takes two arguments—a list of patterns that make up the query and a list of variable bindings. *Deduce* augments the list of bindings it is given initially with bindings for variables in the pattern list and returns a stream of augmented variable bindings. (I will explain streams later; for now you can think of them as lists.) From these augmented bindings, the caller of *deduce* can obtain values for the variables in the query. The pattern list is taken as a conjunction, so if called with the list *((in *x *y) (in *y *z))*, *deduce* will return a stream of all bindings for **x*, **y*, and **z* that satisfy both patterns.

Deduce begins by checking to see if the list of patterns is empty; if so, the query is trivially true and *deduce* returns a single-element stream consisting of the current bindings. Again, you can think of stream operations such as *stream-cons* as their list equivalents, and you can treat **empty-stream** as nil.

If the list of patterns is not empty, *deduce* first obtains a stream of bindings statisifying the first pattern, using the *deduce-pattern* function. Then it calls itself recursively on the remaining patterns for each binding in that stream, appending all the resulting binding streams into one large stream (that is the function of *stream-mapcan*).

Deduce-pattern simply calls *deduce-pat* with the pattern, bindings, and a list of facts and rules from the database that might possibly unify with the pattern. In its simplest form, *find-possible-unifiers* could just return the entire database all the time; I will make use of a slightly more sophisticated indexing scheme that can cut down significantly on the number of facts and rules returned.

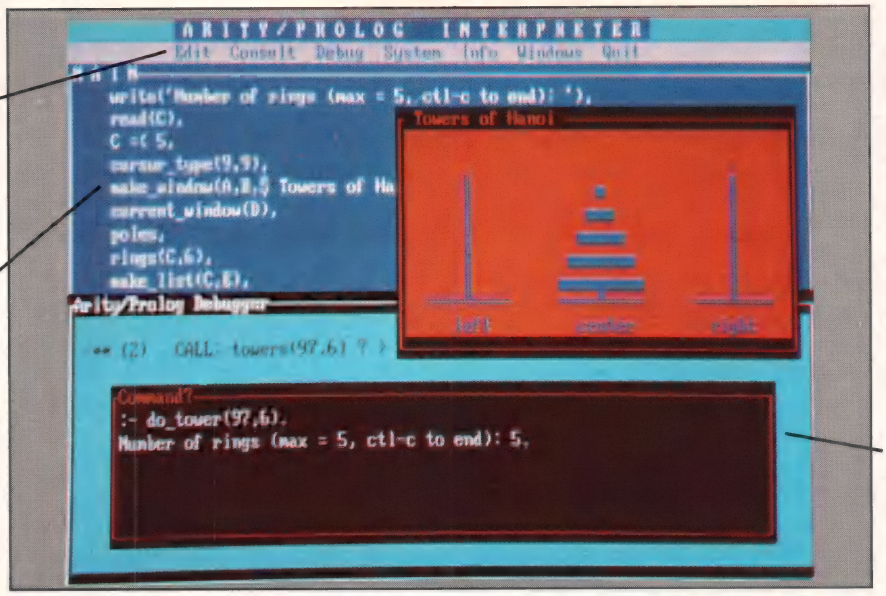
Deduce-pat takes a pattern, a list of bindings, and a list of possible unifiers and returns a stream of augmented binding lists. It first checks to see if there are any more possibilities. If not, it returns the empty stream, indicating failure. Otherwise, it takes the first possibility and tries to unify it with the pattern, renaming its variables first if necessary.

If the unification fails, *deduce-pat* calls itself recursively on the remaining possibilities. If the unification succeeds, deduce-pat calls deduce on the antecedents of the rule (which will be null if the rule is actually a fact) and the bindings produced by the unification. It appends the resulting stream to the stream obtained from calling itself recursively on the remaining possibilities and returns the result. This call *to stream-append*, like the earlier call to *stream-mapcan*, is necessary to ensure

The right tools make your job easier

Built-in full-screen program editor.

Faster, more efficient interpreter. Full superset of the Edinburgh Prolog standard featuring over 200 predicates designed to meet your programming needs.



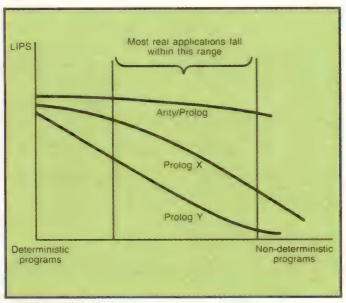
BONUS: Full set of screen design predicates so you can include windows, menus, dialog boxes, and edit boxes like these in your applications. A unique message-passing architecture gives you complete control over the placement, color, and actions of the screen design elements.

Interactive, multiple-window debugger.

Introducing Arity/Prolog Version 5
The right tool for software development

Arity/Prolog Version 5 combines the versatile Prolog language with a fully-integrated user interface and comprehensive language extensions. The result is a fast, full-featured development environment designed to make your job easier.

Speed — when it counts



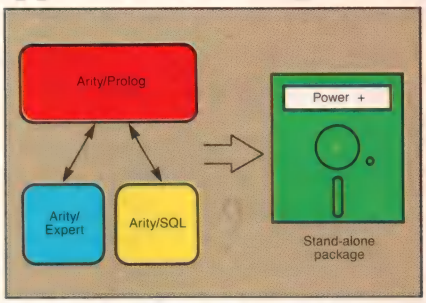
Arity/Prolog V5 sets new standards in Prolog benchmark speeds. Such as 15530 LIPS for Naive Reverse running on an 8 MHz Compaq Portable 286™. But your programs rarely resemble simplistic benchmarks. That's why Arity/Prolog is designed for optimum performance based on typical programming tasks. While the performance of other Prologs declines sharply when tested with real applications, Arity/Prolog's performance remains consistently high.

C, Pascal Language integration — not just interfaces

```
-public count_it/2:c('_count_it'(int,*int)).

count_it(Init,CountPtr):-
        count(init,Count),
        compute(int=*Countptr,
                *Countptr:=count).

count(100,_):-!,fail.
count(X,Y):-
        inc(X,Y).
count(X,Y):-
        count(X,Y)
```

You don't have to give up familiar tools, like C and Pascal, to take advantage of the power of Arity/Prolog. All C data types and expressions are integrated into Arity/Prolog V5. So you have a choice of using your existing C, Pascal, Fortran, or assembly code with Arity/Prolog V5, or including the declarative programming tasks as part of your Prolog program. And now you can call Arity/Prolog from your C function as well.

Part of a family of application building blocks



Arity also offers **Arity/Expert,** a powerful and flexible expert system development package, and **Arity/SQL,** an ANSI standard implementation of the Structured Query Language for use with the Prolog database. Each of these products is closely integrated with Arity/Prolog V5, making it easy to build customized applications.

That's just the beginning

A R I T Y

Arity Corporation
30 Domino Drive
Concord, Massachusetts
01742

Arity/Prolog V5 includes many more features, such as a virtual database which supports up to 1 gigabyte, database partitioning and indexing, DCG support, and string and floating point support. Arity/Prolog runs on IBM PCs and compatibles. Call today to get more information.

1-800-PC-ARITY
(Mass: 617-371-1243)

Compaq Portable 286 is a trademark of Compaq Computer Corp.
IBM is a registered trademark of International Business Machines Corp.

Arity/Prolog Compiler and Interpreter ■ Arity/Expert Development Package ■ Arity/SQL Development Package

CIRCLE NO. 112 ON READER SERVICE CARD

(continued from page 24)

that every possible binding of the query variables is found.

Several internal functions of the deducer deserve mention. The *unify* function takes two patterns and a set of bindings and tries to unify the patterns. It returns the bindings augmented with new ones if it can unify and the symbol *:fail* if it can't. It proceeds by recursively *cdr*ing down the patterns, trying to match each element. Variables are handled by *unify-var*: unbound variables are bound and the match succeeds; bound variables are treated like their values, by calling *unify-const*. *Unify-const* just compares the two elements for equality, returning the existing bindings if they are equal and *:fail* if not. This unifier does not handle patterns containing nested structures as does PROLOG's, but it could easily be modified to do so.

Variable bindings are implemented as LISP association lists (alists). The *add-binding* function just *cons*es a new pair onto the binding list, and the *var-value* function uses LISP's built-in *assoc* function repeatedly to find the value at the end of the chain of bound variables.

The functions *add-to-database*, *remove-from-database*, and *find-possible-unifiers* implement the database indexing scheme. This scheme is quite simple: facts are stored by their first element, on the element's property list. For instance, all the facts beginning with *in* are stored together. Backward rules (the only kind of rule stored in the database) are stored by the first element of their consequent, unless that element is a variable, in which case the rule is stored under the * symbol. The possible unifiers for a pattern whose first symbol is a constant include those on the property list of the constant and the rules stored under *. If the first element of a pattern is a variable, you are forced to search the whole database.

This indexing scheme has the advantage of generality. Further improvements could be obtained, still preserving generality, by indexing off more than the first element; see note 8, Chapter 14, for one such technique. Because many relations, such as *in*, will be ubiquitous in adventure games, they can be treated specially to improve efficiency even more. For instance, if the program maintains, with each object, a list of things that that object contains, then finding the values for *x in the pattern (*in *x object*) is just a matter of fetching that list. Many such optimizations are possible, but keep in mind that they are efficiency hacks, to be hidden in the database indexing mechanism and not made a feature of the language. The view of the adventure world as a single database of facts is a great strength of AAL.

Streams

If streams are just lists, then *deduce* will find every possible binding of the variables and return a list of all of them. This could be a big waste of time if you want only the first binding. And if (as is possible) there are an infinite number of answers, then *deduce* will never return. It would be nice if *deduce* worked more like PROLOG, returning the first answer as soon as it is found but remembering where it was so that subsequent answers could be requested. In fact, this is just how *deduce* works because streams are not lists but special data structures with a rather interesting property.

Streams behave much like lists—they have a *car*, which is a value, and a *cdr*, which is another stream (possibly the empty stream). They are constructed using an operation such as *cons*. The crucial difference is that a stream's elements are not all evaluated; only the first is. The evaluation of the remaining elements is delayed until they are actually requested.

Compare the two expressions:

```
(cons 1 (cons 2 nil))
```

and:

```
(stream-cons 1 (stream-cons 2 *empty-stream*))
```

In the first of these, the inner call to *cons* is evaluated, then the outer one, resulting in the list *(1 2)*. But in the second expression, the inner call to *stream-cons* is not evaluated immediately. Its evaluation is delayed, and the resulting object looks something like:

```
(1 [delayed computation of (stream-cons 2 *empty-stream*)])
```

(This notation is merely for explanation; you'll see the actual implementation in a moment.) When *stream-car* is called on this object, *1* is returned immediately. But calling *stream-cdr* forces the evaluation of the delayed computation. This results in a new stream, which you can write as:

```
(2 [delayed computation of *empty-stream*])
```

Another call to *stream-car* will produce *2*, as it should, and calling *stream-cdr* again will result in the empty stream. So, using *stream-car* and *stream-cdr* on a stream is just like using *car* and *cdr* on a list, except for when the work is done.

The implementation of streams is remarkably simple, provided you have a lexically scoped LISP such as Scheme or Common LISP (see Listing Three, page 62).

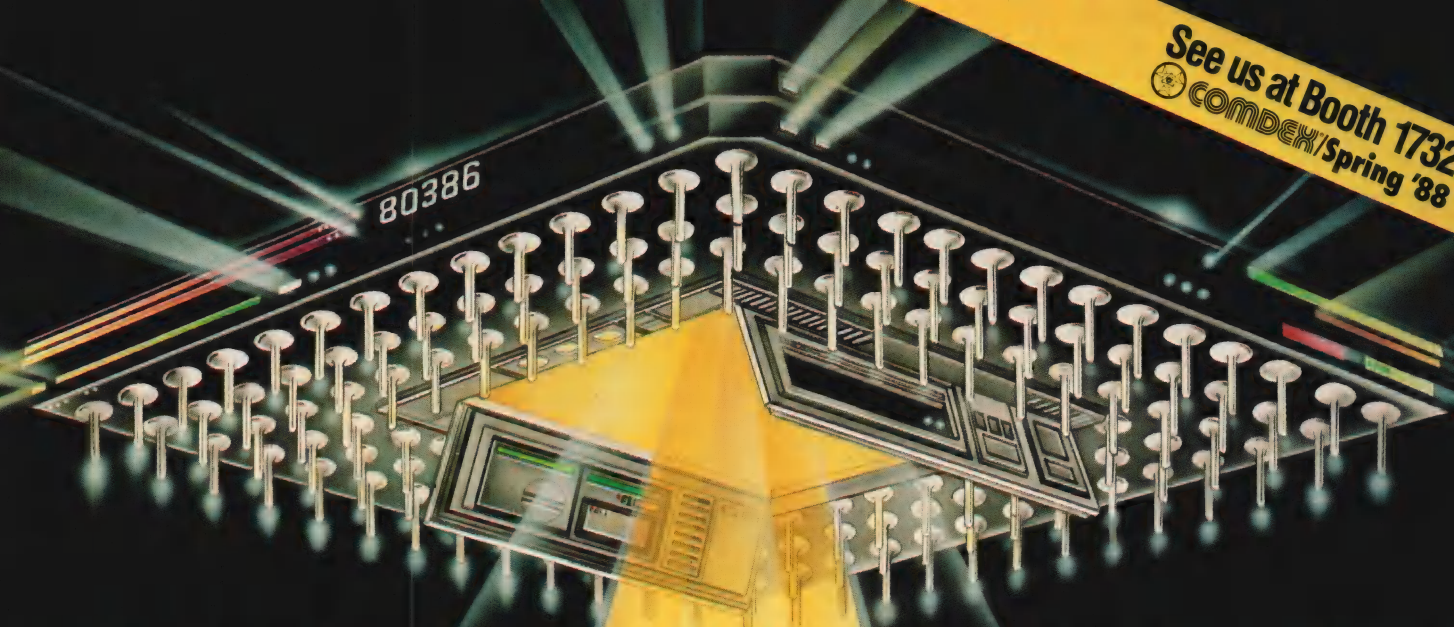
You can implement them in terms of two primitives—*delay* and *force*. *Delay* takes a LISP expression and turns it into a delayed computation; *force* takes a delayed computation and evaluates it.

To delay a computation, all you have to do is enclose it in a function of no arguments, so an expression such as *(+ 2 3)* becomes *#'(lambda () (+ 2 3))* (in Scheme, the # and ' characters aren't necessary). You can write *delay* as a simple macro:

```
(defmacro delay (thing)
  `#'(lambda () ,thing))
```

To force a delayed computation, you need only funcall it, so you can write *force* like so:

See us at Booth 1732
COMDEX/Spring '88



UP TO 25 USERS.

MADE FOR THE 80386.

RUNS DOS PROGRAMS.

MULTI-TASKING

Imagine the speed and power of a $100,000 minicomputer in a desktop PC costing under $7,000. Now imagine all that power going to waste because the operating system you chose was never meant to take advantage of a computer this powerful. It will take more than just a "window environment" or an outdated operating system to unlock the 80386.

It will take PC-MOS/386.™

The First 80386 Operating System. Specifically designed for the 80386 computer, PC-MOS/386™ opens doors. Doors to more memory and multi-tasking. Doors to thousands of DOS programs as well as upcoming 80386-specific software. It's the *gateway* to the latest technology... and your networking future.

Memory Management Without Boards. PC-MOS exploits the memory management capabilities built into the 80386. So, up to four GIGABYTES of memory are accessible to multiple users and to future 80386-specific applications requiring megabytes of memory.

Multi-Tasking, Multi-User Support for One, Five or 25 Users. PC-MOS/386™ allows up to 25 inexpensive terminals to be driven by a single 80386 machine. So the features of the 80386 can be utilized at every terminal. And it comes in three versions so you can upgrade your system as your company grows...without having to learn new commands or install new hardware.

Software Support for Thousands of DOS Programs. Although PC-MOS/386™ totally replaces DOS, it doesn't make you replace your favorite DOS programs. So you can run programs like Lotus 1-2-3, WordStar, dBASE III, and WordPerfect on the 80386. Best of all, it uses familiar commands like DIR and COPY-so you'll feel comfortable with our system.

The Gateway to Endless Features. Distinctive characteristics like file/system security, remote access, file/record locking, and built-in color graphics support for EACH user set PC-MOS/386™ apart from all previous operating systems.

Open the Doors to Your Future TODAY! Call The Software Link TODAY for more information and the authorized dealer nearest you. PC-MOS/386™ comes in single, five & 25-user versions starting at $195.

PC-MOS/386™
MODULAR OPERATING SYSTEM

THE SOFTWARE LINK
Developers of LANLink™ & MultiLink® Advanced

3577 Parkway Lane, Atlanta, GA 30092
Telex 4996147 SWLINK
FAX 404/263-6474

for the dealer nearest you.
CALL: 800/451-LINK
In Georgia: 404/448-LINK

OEM/Int'l Sales: 404/263-1006
Resellers/VARs: 404/448-5465

OEM/Dealer Inquiries Invited

THE SOFTWARE LINK/CANADA CALL: 800/387-0453

CIRCLE NO. 113 ON READER SERVICE CARD

More Than Just Windows, We've Opened Doors.

(continued from page 26)

```
(defun force (thing)
  (funcall thing))
```

You can implement a stream as a LISP dotted pair (*cons* cell) whose *car* contains the *car* of the stream and whose *cdr* contains the delayed computation. Constructing a stream from a value *v* and a computation *c* can be done by writing *(cons v (delay c))*, which is just what *stream-cons* does:

```
(defmacro stream-cons (thing comp)
  `(cons ,thing (delay ,comp)))
```

Note that *stream-cons* must be a macro so that the second argument is not evaluated.

The other stream functions are simple. *Stream-car* is identical to *car*, whereas *stream-cdr* has to force the *cdr* of the stream object. You can represent the empty stream with nil, making the *stream-empty?* predicate equivalent to null. With these functions in place, you can write more complex ones such as *stream-mapcan* and *stream-append* just as you'd write their equivalent list-manipulating versions. The remaining trick is to delay the second argument of *stream-append* so that only the first argument is evaluated.

What is the effect of using streams in *deduce*? Instead of computing all the answers before returning any, *deduce* will take the first answer it finds and return a stream whose *car* is that answer and whose *cdr* represents the rest of *deduce*'s work. If only the first answer is desired, the rest of the stream can be thrown away and no more work will be done. If another answer is wanted, calling *stream-cdr* on the stream will generate it. In this way, only as many answers as required are actually computed.

Streams have many useful applications besides this one. For an excellent description of streams and their uses, including a PROLOG-like interpreter similar to the one I've presented here, see note 7.

Applications of the Deducer

Let's consider three uses of the deducer in the implementation of AAL: rules, the *every* action, and requirements checking.

Much of the work in a typical AAL program is done by rules. A typical rule might say:

```
((at lamp *x) (on lamp) -> (lit *x))
```

To evaluate the rule, you first see if the antecedents (the patterns to the left of the -> symbol) can be satisfied, and if so you execute the actions on the right of the -> with the bindings for the variables on the left. Only the first set of bindings that matches the antecedents is used, so you really do not need the full power of streams here. The implementation is simple:

```
(let ((bindings-stream (deduce (rule-antecedents rule)
                               bindings)))
  (if (stream-empty? bindings-stream)
      :did-not-fire
      (do-rule-actions (rule-consequents rule)
                       (stream-car bindings-stream))))
```

Here, bindings are whatever bindings are in force at the time. If the rule occurs at top level, these will be the bindings of the AAL global variables. But rules may also occur as actions in other rules, so the bindings may contain variables accumulated from those rules.

As another application of the deducer, consider a kind of AAL action I haven't yet mentioned—the *every* action. It allows an action to be taken for every possible binding of a variable. For instance, to "take inventory"—that is, display the items that the player is carrying—you could write:

```
(every *x (carrying player *x) -> (lisp (print *x)))
```

Implementing the *every* action is trickier than it might seem. The basic idea is to obtain the stream of bindings from using *deduce* on the antecedents, then execute the consequents for each binding. The problem with this approach is that a binding may be repeated if the deducer can derive it via different routes. The solution is to turn the stream of bindings into a list, then remove the duplicates. The implementation is shown in Listing Four, page 62.

As my final and most complex example, consider how the requirements for an action are checked. The compiler parses each *requires* specification into a list of requirement structures, each of which has three fields: a pattern to be checked against the database; a string to output if the pattern fails; and a Boolean variable called *succeeded?*, whose use I'll describe later. These structures (along with everything else mentioned when an AAL entity is defined) are stored on the property list of the entity's identifier.

Requirement checking is done by the *satisfies-requirements* function, which is called with the command, object, and instrument of the user's command and calls *check-requirements* for each of the three. *Check-requirements* gets the list of requirements for the command from the given entity, sets their *succeeded?* fields to nil, then calls *check-reqs* with the requirements list and the list of bindings of AAL's global variables. If *check-reqs* returns *t*, so does *check-requirements*; otherwise, *check-requirements* displays the string returned by *check-reqs* and returns nil.

Check-reqs' job is to return *t* if all the requirements can be satisfied or the string that should be printed if they can't all be satisfied. If AAL didn't allow different strings to be associated with each pattern, then you could represent the requirements as a list of patterns and implement *check-reqs* very easily using *deduce*:

```
(defun check-reqs (reqs bindings)
  (if (stream-empty? (deduce reqs bindings))
      "The requirements could not be satisfied"
      t))
```

But the presence of strings for each pattern doesn't

Discover a new world of C performance.

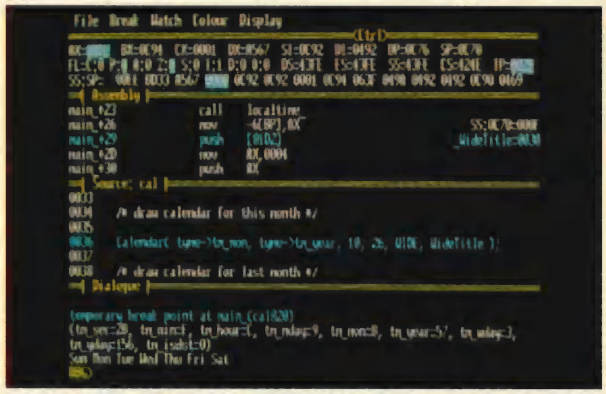
IMMEDIATE DELIVERY!
1-800-265-4555

At a special low introductory price!

WATCOM announces a new team of high-performance C language development systems that deliver proven superior results. Both are available now, at low introductory prices, for IBM PCs, PS/2s, and compatibles.

Best of Both Worlds.

Both systems are optimizers. Express C optimizes your time, WATCOM C6.0 optimizes your code. You win both ways!



With the WATCOM VIDEO debugger you can debug large applications without extended memory.

System Requirements

- System: IBM PC, PC XT, PC AT, PS/2, or true compatible
- Recommended memory: 512K
- Operating system: PC-DOS or MS-DOS, version 2.0 or later.

WATCOM C6.0 Optimizing Compiler and Tools For the Fastest Tightest Code.

This unique development system produces the fastest execution speeds and smallest code available, as shown in benchmark tests against Microsoft C5.0 and Turbo C. It includes the new WATCOM VIDEO Debugger which quickly diagnoses elusive bugs without the need for extended memory even in very large programs. WATCOM C6.0 comes with a copy of Express C and offers a broad spectrum of advantages including: Extensive fine-tuning capabilities. A sophisticated register allocation scheme that eliminates many costly memory references. True register variables. Flow analysis. Altogether it allows your code to run its quickest.

Without a doubt, WATCOM C6.0 is the ideal choice for all memory models, small to huge, and on systems with or without 80x87. Its flexible run-time conventions also allow efficient interfacing with a wide range of libraries and language processors.

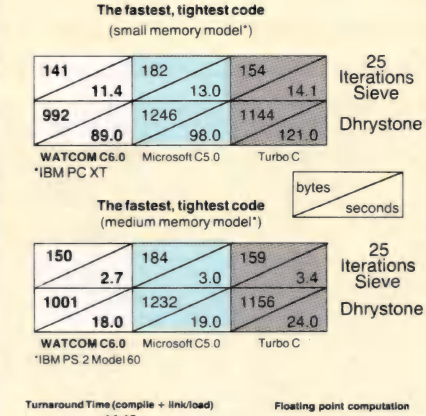
Superlative Performance

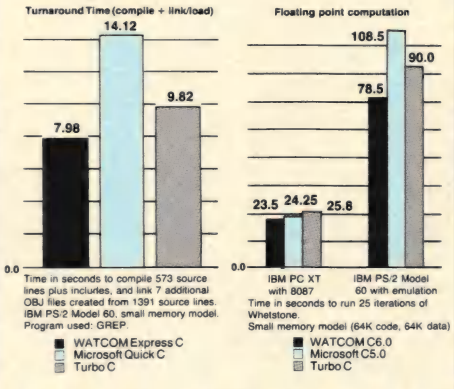
- ☑ Full ANSI C Optimizing Compiler
- ☑ Visual Interactive Debugger
- ☑ Full ANSI C Run-time Library
- ☑ Source Editor
- ☑ WATCOM C and Express C User's Guides
- ☑ WATCOM C Language and Library References
- ☑ WATCOM Editor User's Guide
- ☑ On-line Help Text
- ☑ Disassembler
- ☑ Overlay Linker
- ☑ Object Librarian
- ☑ MAKE
- ☑ **Express C**

List Price: $495
Introductory Price: **$295***

Software Credentials

WATCOM C6.0 is the product of 20 years of computer language experience dating back to the creation of WATFOR in 1965. Our commitment to technical support matches our commitment to deliver the world's fastest and most productive programming tools. With more than 400,000 software products in worldwide use and site licensing available for multiple machines and networks, you simply cannot find a better source of software development tools.





WATCOM Express C For the Fastest Development Environment.

This seamless development environment offers high speed compilation and the ultimate in programming ease. It is an integrated editor, compiler, debugger, linker and run-time system. With unexcelled diagnostic capabilities, it quickly checks apparently correct code and uncovers common or difficult bugs that other compilers miss. Express C provides you with reliable code and exceptional productivity.

Unparalleled Productivity

- ☑ Full ANSI C Compiler
- ☑ Integrated Source Editor
- ☑ Integrated Debugger
- ☑ Full ANSI C Run-time Library
- ☑ Integrated linker/loader
- ☑ On-line Help Text
- ☑ WATCOM C Library Reference
- ☑ WATCOM Express C User's Guide
- ☑ WATCOM C Language Reference
- ☑ Overlay Linker
- ☑ Object Librarian
- ☑ MAKE

List Price: $125
Introductory Price: **$75***

SPECIAL INTRODUCTORY OFFER!
___ C6.0 at introductory price $295
___ Express C, at introductory price $75
___ Please send product information.
___ Please contact re: site licensing and corporate price quotes.

*Limited time introductory prices apply only to prepaid orders. (VISA/MasterCard)
Shipping and handling extra.

WATCOM

Dept. DD-04B, Suite 306-21
1430 Massachusetts Ave., Cambridge, MA 02138

Available now. For immediate delivery in the USA and Canada Call:

1-800-265-4555

Name _____
Title _____ Tel. # _____
Company _____
Street _____
City _____ State ___ Zip ___
Visa __ MasterCard __ Card # _____
Exp. Date ___ Signature _____

CIRCLE NO. 114 ON READER SERVICE CARD

DESQview API Reference Manual

This is the primary source of information about the DESQview API. It contains all you need to know to write assembly language programs that take full advantage of DESQview's capabilities. The Reference manual comes with an include file containing symbols and macros to aid you in development. AVAILABLE NOW!

DESQview API C Library

The DESQview API C Library provides C Language interfaces for the entire set of API functions. It supports the Lattice C, Metaware C, Microsoft C, and Turbo C compilers for all memory models. Included with the C Library package is a copy of the API Reference Manual and source code for the library. AVAILABLE NOW!

DESQview API Debugger

The DESQview API Debugger is an interactive tool that enables the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter, registers, and stack at the time of the call. Trace conditions can be specified so that only those calls of interest are reported. AVAILABLE JUNE 88

DESQview
Quarterdeck

Introducing DESQview 2.0 API Tools

Bringing new power to DOS

DESQview API Panel Designer

The DESQview API Panel Designer is an interactive tool to aid you in designing windows, menus, help screens, error messages, and forms. It includes an editor that lets you construct an image of your panel using simple commands to enter, edit, copy, and move text as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel, such as its position, size, and title. Finally, you can specify the locations and types of fields in the panel.

The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped together into panel libraries and stored on disk or as part of your program. AVAILABLE JUNE 88

DESQview API Pulldown Menu Manager

The DESQview API Pulldown Menu Manager is an interactive tool to aid you in designing pulldown menus. This DESQview API tool assists you in giving your DOS program an OS/2-like look and feel. AVAILABLE JULY 88

Quarterdeck Office Systems
150 Pico Boulevard
Santa Monica, CA 90405
(213) 392-9851

(continued from page 28)

permit this approach because, when *deduce* returns an empty stream, you don't know which pattern was responsible for the failure. You need to use *deduce-pattern* on each pattern individually. One plausible first attempt might look like Example 1, below. Unfortunately, this isn't correct. Because of backtracking, it is a little tricky to pin down just which requirement is to blame when failure occurs.

Consider the requirements of lighting a lamp, which stipulate that the lamp contain batteries that aren't dead:

```
(requires ((in *x lamp) "Nothing is in the lamp")
          ((battery *x) "There are no batteries in the
                                                  lamp")
          ((not (dead *x)) "The batteries are dead"))
```

You would like a message to print out only when the corresponding pattern is the one responsible for failure. But the guilty pattern is not always the last pattern that fails.

Consider a situation in which there are two things in the lamp, only one of which is a battery, and the battery is dead. Say that when *deduce-pattern* is called with the first pattern *(in *x lamp)*, it returns a stream whose first element is a binding list with **x* bound to the battery. Then the second pattern *(battery *x)* succeeds, but the third one fails because the battery is dead. A message should not be printed out yet because there might be other, nondead batteries in the lamp. So you eventually backtrack to the first pattern and get the second binding for **x*, which is the other object in the lamp. Now the second pattern fails right away because the second object is not a battery. Clearly, you would like to say that the reason why the lamp could not be lit was that the batteries in it were dead; the fault is with the third pattern, not the second. But the version of *check-reqs* in Example 1 will return "There are no batteries in the lamp."

The key to the right solution is to notice that once a pattern has succeeded once, it cannot be the cause for

```
(defun check-reqs (reqs bindings)
  (if (null reqs) ;; all requirements passed--success
   T
   ;; else, check the first
   (let* ((req (car reqs))
          (binding-stream (deduce-pattern (requirement-pattern req)
                                          bindings))
          (result))
     ;; if it failed, return its failure string
     (cond
      ((empty-stream? binding-stream)
       (requirement-failure-string req))
      ;; else, try all the bindings on the other reqs until
                                                  success.
      ;; dostream just iterates over the elements of the stream.
      (t
       (dostream (binds binding-stream)
        (setq result (check-reqs (cdr reqs) binds))
        (if (eq result t)
          ;; the remaining requirements passed--success
          (return-from check-reqs t)))
       ;; No other bindings worked; return the last failure string
       result)))))
```

Example 1: *An incorrect attempt to use deduce-pattern on each pattern individually*

KEEP UP WITH THE OS/'s

Megabytes of Memory and 32-Bit Performance for DOS.

If you thought the only way to protected mode was the big move to OS/2...We have good news! You can gain the benefits of protected mode the easy way with OS/286™ and OS/386™. These tools for C, Fortran, Pascal and Assembly language programmers permit rapid conversion of existing DOS applications from "real" 8086 mode to "protected" 286 and 386 mode. They don't replace or modify DOS, but extend it to protected mode.

OS/286 and OS/386 are the only DOS extenders that span both the 286 and 386 processors, with 32-bit capability *today* on 386s that yields twice the performance of 16-bit mode. OS/286 and OS/386 have quickly become the preferred solution for developers of high performance, memory-intensive applications, including CADKEY, CASE, and Gold Hill, and premier language developers Lahey, and Metaware.

Our optional TOUCHDOWN™ BIOS supplement provides fast and reliable protected mode operation on *any* 286 system, even those with problems resetting the 286. (Ever notice how few existing machines Operating System/2 runs on?)

If your applications are running out of memory or need more speed, don't wait for the "solution" that means abandoning your investment in DOS. Enhance them now with OS/286 and OS/386.

OS/286™ & OS/386™ Benefits:

- Gain multi-megabytes of directly addressable memory (15-Mb-286, 4Gb-386)
- Increase performance by eliminating overlays and EMS
- Convert in days, not months
- Continue to work with a DOS interface and use existing TSRs, device drivers, graphic routines, etc.
- Stay compatible with the widest array of systems

A.I. Architects Software Developers Kit **$495**

includes full support for:

- MetaWare High C (16 & 32 bit)
- Professional Pascal (16 & 32 bit)
- Lahey F77L FORTRAN
- Microsoft C & Fortran 4.0,
- MASM, MS-Link
- Phoenix PLINK86
- Halo & GSS Graphics
- Pharlap 386: ASM/LINK

more to come

Run time licenses for OS/286 and OS/386 are available at nominal cost.

The HummingBoard™ turns any XT or AT into the fastest 386 system available. The dual processor architecture boosts performance significantly over comparable single processor systems or accelerator boards. Available with 2 to 24Mb RAM, 16 or 20Mhz speed, and 387 floating point coprocessor.

A.I. Architects, Inc.

One Kendall Square, Cambridge, MA 02139
TEL (617) 577-8052 FAX (617) 577-9774

(continued from page 32)

ultimate failure, even if it later fails. Only a pattern that never succeeds can be guilty. The correct version of *check-reqs* uses the *succeeded?* field of the requirement structure to record this fact; it is given in Listing Five, page 63.

Review

You've seen how AAL combines three programming styles into a powerful language. At AAL's heart is the deductive-retrieval or logic-programming style of PROLOG; AAL represents the state of an adventure game as a database of facts, and AAL programs work by examining and manipulating this database. The pattern-matching, rule-following, and backtracking abilities of the deducer make it easy to write powerful rules and conditions simply.

AAL uses the idea of inheritance from object-oriented languages to ease the definition of game objects, locations, and commands. Each of these entities can possess one or many features, which not only act as abbreviations for properties but also serve as predicates in the database. Features can take arguments and may themselves have features, allowing complex networks to be constructed. Using features to group commands, locations, and objects can also result in considerable economy in rule writing because a whole group of entities (for example, commands that move the player) can be described with a single pattern.

Finally, AAL would have been an order of magnitude more difficult to implement were it not written in and on top of LISP. LISP's reader and macro facility trivialized the parser and compiler for AAL. LISP's property lists, built-in symbol table, and support for association lists simplified many aspects of the implementation. Its ability to construct and call functions on the fly made possible the implementation of the vital *stream* data type. And by allowing AAL programs to escape to the underlying LISP system, I obtained hundreds of useful functions for free.

What Next?

Here I have offered only a sketch of AAL, which is itself only a small part of what could be done in this direction. Many of the fine points of AAL and its implementation can be gleaned from reading the source code. But the deducer stands on its own as a useful program, or it can serve as the beginning of a PROLOG implementation. I have taken care to have all the essential code for the deducer published with this article, so you can begin without delay.

I realize that Common LISP is, despite its name, not terribly common compared to languages such as C and Pascal. But other LISPs, such as XLISP and Scheme, are available and will do just as well. And it is certainly possible to translate AAL into a language such as C, though I would guess the code size would more than double. The major difficulty concerns streams, which cannot be implemented in their full generality in a language that does not allow run-time creation of func-

Object-Oriented Engineering

If you don't pick the right environment, you may come up dry.

You've probably heard about object-oriented engineering. And the advantages it brings to software development.

You may even be ready to dive into it. We suggest you look before you leap.

Many companies today only offer one small bit of the support you need.

Not Stepstone. We give you more than just a compiler. We supply a whole object-oriented environment complete with powerful interpreter/debugger.

So you can shorten development schedules. Reduce maintenance costs. Build code which directly models your problem domain. And build truly re-usable software components.

Our products are based on the well known Objective-C® language and are designed for programmers who want the benefits of this new technology without giving up the advantages of C.

We even offer more than just a development environment.

We sell libraries of powerful, robust classes you can use directly in your applications. Like ICpak™ 201, a set of pretested Software-IC®s for building custom graphical user interfaces.

Our products operate in a wide variety of work-station environments. As well as on the PC-AT™ and compatibles.

In addition, our support is second to none. We design, build, market and service our own products. Hence, we will do all that's necessary to keep you completely satisfied.

So before you jump into object-oriented engineering, talk to the company that's got the fullest range of products. Stepstone.

We won't let you down.

Stepstone™
The Leader in Object-Oriented Technology

The Stepstone Corporation 75 Glen Road Sandy Hook, CT 06482 203 426 1875 Telex 506127 FAX 203 270 0106

PC-AT is a trademark of International Business Machines Corporation. Objective-C and Software-IC are registered trademarks of The Stepstone Corporation. Stepstone and ICpak are trademarks of The Stepstone Corporation.

CIRCLE NO. 122 ON READER SERVICE CARD

Our front end helps protect your back end.

Today's users require sophisticated interfaces for their applications. Yet complex front ends are a real pain to create, especially when the specs change and your deadlines don't.

But now JYACC introduces JAM™, a powerful user interface development tool that makes it easier than ever to design and revise your complex applications.

Use windows and colors freely with JAM.

JAM is the first tool that does it all, from prototyping to implementation. With JAM, you start by creating screens and linking them together to develop an application shell. You can experiment with the look of the interface, and even explore "what if" scenarios on the application flow. Then you attach processing routines, and your application is complete.

JAM works under the following operating systems:

- UNIX®
- MS-DOS®
- VMS®
- XENIX®
- RMX™
- VOS™

You'll be amazed at how quickly your applications spring to life with JAM's interactive screen management utility. You can use features like context-sensitive help, shifting and scrolling fields, a variety of visual attributes and extensive data validations to create exciting screens, windows and menus—all without writing a single line of code. JAM also lets you test your prototypes at any time.

JAM lets you draw from its extensive subroutine library to help you write processing routines faster. And revisions to your applications are easier, because your subroutines are insulated from the data entry and presentation details of the interface.

JAM is extremely portable. It runs on almost every computer, from PCs to superminis, and works under 6 operating systems. This allows you to develop consistent interfaces throughout your company—a significant asset to the Fortune 500 companies that have been using JAM for more than a year.

Enhance applications with context-sensitive help.

JAM from JYACC. It gets your front ends—and back ends—in great shape. Call for more information about JAM and our demo diskette. 800-458-3313

JYACC FORMAKER™, JAM's screen manager, is also available separately.

JYACC, Inc., 116 John Street
New York, NY 10038 212-267-7722

Introducing
JAM

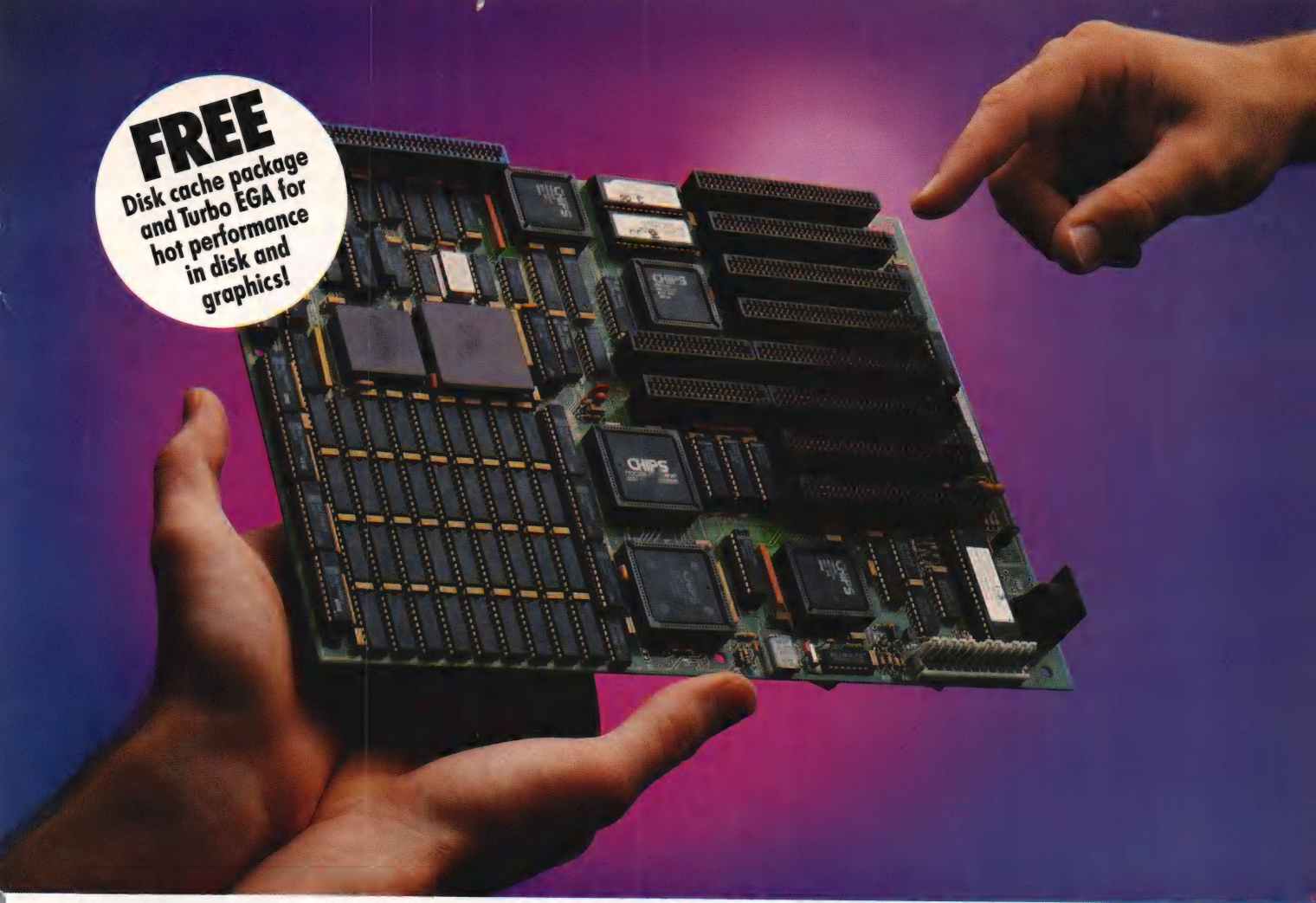
JYACC Application Manager. *The Composer for Sophisticated Applications.*

JYACC

Excellence in Systems & Application Design

MS-DOS, XENIX: Microsoft Corp.; UNIX: AT&T Bell Labs; RMX: Intel Corp.; VMS: Digital Equipment Corp.; VOS: Stratus Corp.

CIRCLE NO. 123 ON READER SERVICE CARD



CONVERT YOUR PC, XT OR AT INTO A HIGHER FORM OF LIFE!

386 MOTHERBOARDS FOR 386 SPEED

Don't let your PC give up the ghost—Hauppauge has just arrived with a new spark of life: the 386 MotherBoard.™ Far more advanced than an accelerator card, our line of MotherBoards grace your PC, PC/XT, PC/AT or compatible with speeds equal to the IBM PS2 Model 80. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that make users humble with awe.

OS/2 Compatible. To ensure a long, fruitful life, our 386 MotherBoard is compatible with the PC/AT (BIOS and I/O)—so you can run the new generation of DOS, OS/2. Our Board also runs Windows/386, UNIX V and PC-MOS/386. For more power, you'll find 16-bit expansion slots that accommodate the latst I/O expansion cards. No 386 accelerator card gives you so much versatility. **Only our 386 MotherBoard gives your PC a future with unlimited possibilities!**

The Critics Applaude! *PC Magazine* awarded our Board "The Editor's Choice" for 386 Replacement Boards. *PC World* called it "the Upgrade Product of the Year."

Technical Features ▪ 16 MHZ 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Six 8-bit expansion slots ▪ two 16-bit expansion slots (four on 386 MotherBoard/AT) ▪ One 32-bit expansion slot for up to 12 Megabytes of high speed memory ▪ Battery-powered clock calendar

386 MotherBoard/PC or MotherBoard/XT	$1495
386 MotherBoard/AT	$1595
32-bit RAM Board (2 Mbytes installed; up to 4 Mbytes)	$795
16 MHz 80387 math coprocessor	$695
16-bit combination hard disk/ floppy disk controller	$245

For more information on our easy-to-install MotherBoard, call: **1 (800) 443-6284**. In New York, call **(516) 434-1600**.

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788

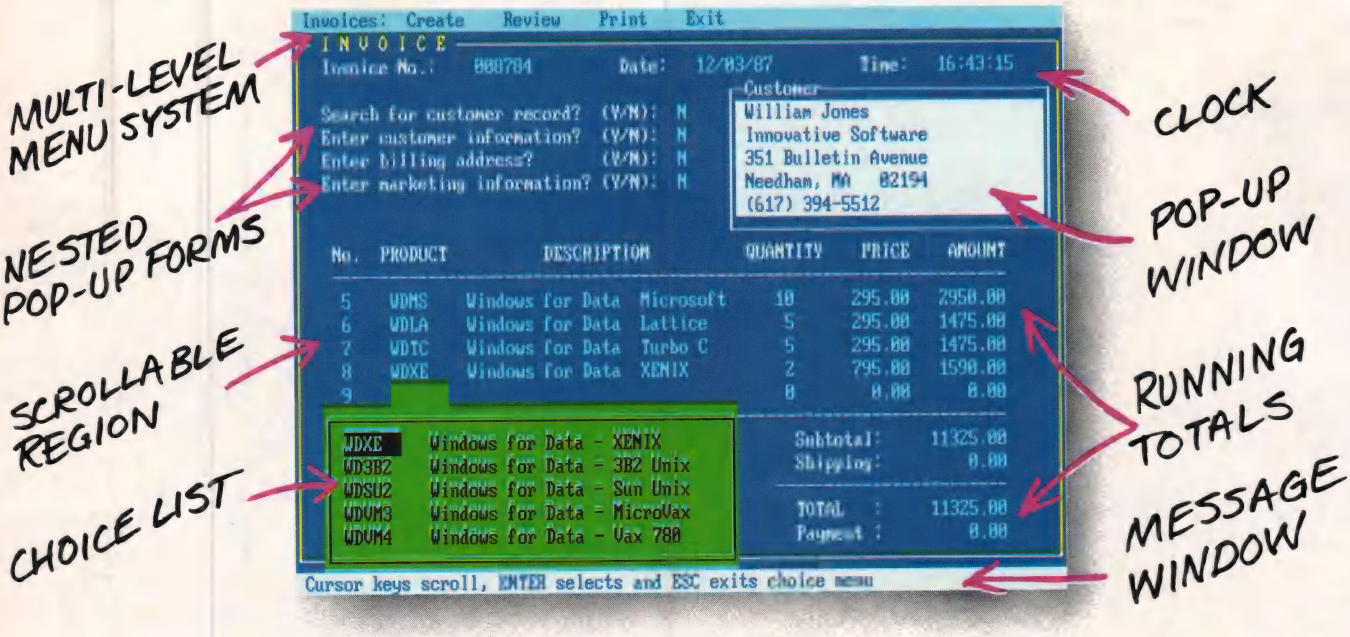
Hauppauge!

CIRCLE NO. 124 ON READER SERVICE CARD

New Prices

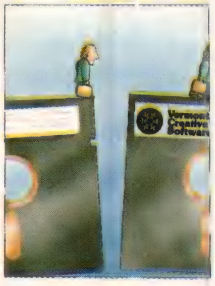
OS/2

WINDOWS FOR DATA®



If you program in C, take a few moments to learn how Windows for Data can help you build a state-of-the-art user interface.

- ☑ Create and manage menus, data-entry forms, context-sensitive help, and text displays — all within windows.
- ☑ Develop window-based OS/2 programs right now, without the headaches of learning OS/2 screen management. Run the same source code in PCDOS and OS/2 protected mode.
- ☑ Build a better front end for any DBMS that has a C-language interface (most popular ones do).



FROM END TO BEGINNING

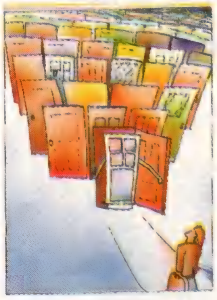
Windows for Data begins where other screen packages end, with special features like nested pop-up forms and menus, field entry from lists of choices, scrollable regions for the entry of variable numbers of line items, and an exclusive built-in debugging system.

NO WALLS

If you've been frustrated by the limitations of other screen utilities, don't be discouraged. You won't run into walls with Windows for Data. Our customers repeatedly tell us how they've used our system in ways we never imagined — but which we anticipated by designing Windows for Data for unprecedented adapatability. You will be amazed at what you can do with Windows for Data.

YOU ARE ALWAYS IN CHARGE

Control functions that you write and attach to fields and/or keys can read, compare, validate, and change the data values in all fields of the form. Upon entry or exit from any field, control functions can call up subsidiary forms and menus, change the active field, exit or abort the form, perform almost any task you can imagine.



OUR WINDOWS WILL OPEN DOORS

Our windows will open doors to new markets for your software. High-performance, source-code-compatible versions of Windows for Data are now available for **PCDOS, OS/2, XENIX, UNIX,** and **VMS.** PCDOS versions are fully compatible with Microsoft Windows. **No royalties.**

MONEY BACK GUARANTEE

You owe it to yourself and your programs to try Windows for Data. If not satisfied, you can return it for a full refund.

Prices: PCDOS $295, Source $295. OS/2 $495. XENIX $795. UNIX, VMS, please call.

Call: (802) 848-7731 **ext. 31**
Telex: 510-601-4160 VCSOFT FAX 802-848-3502

Vermont Creative Software

21 Elm Ave.
Richford,
VT 05476

CIRCLE NO. 125 ON READER SERVICE CARD

tions. Ersatz streams designed expressly for their role in the deducer can be implemented, however, by allocating a structure (or record) and storing the relevant local variables in it directly.

I have hardly mentioned the natural-language aspects of AAL because they are secondary to the concerns of the article and are not well developed in any case. Much could be done with existing natural-language technology to improve the parser and the treatment of command execution. I would guess that the conceptual-dependency representation of Roger Schank and his students might prove useful in this domain (see, for example, note 9).

Let me close with a specific, rather ambitious suggestion for improving the natural-language part of AAL. One weakness of the language is that it forces commands to be specified fully: you must say "throw axe at dwarf" instead of just "throw axe," as the original Adventure would let you do, or "blow whistle" instead of just "blow" when the whistle is the only blowable thing you're carrying. Adventure's resolution of the ambiguity was extremely ad hoc.

One reasonable solution when something is omitted from a command would be to look around for a thing that satisfied the requirements of the command and, if there was only one such thing, to use it. That would take care of blowing the whistle, but not killing the dwarf, and not a case in which the player said "drink" and was carrying a bottle of water, but the cap was on the bottle and the bottle was in a locked chest.

A more general mechanism would determine the player's desired action (drink the water in the bottle), construct a plan to achieve it (open the bottle, open the chest—do I have the key?), and execute the plan. It would be a very ambitious programmer indeed who tried to implement this idea; a good solution would probably be worth a Ph.D. thesis. In fact, see Allen's[10] for a start.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Notes

1. Bertram Raphael, "SIR: A Computer Program for Semantic Information Retrieval," Semantic Information Processing, ed. Marvin Minsky (Cambridge, Mass.: MIT Press, 1968): 33-145.

2. Carl Hewitt, "PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot," *Proceedings of the First International Joint Conference on Artificial Intelligence* (*IJCAI*) (1969): 295–301.

3. Gerald Sussman and Drew McDermott, "Why Conniving Is Better Than Planning," MIT AI Lab Memo, no. 255A (1972).

4. Johan de Kleer; Jon Doyle; Guy Steele, Jr.; and Gerald Sussman, "AMORD: Explicit Control of Reasoning," : 345–356.

5. Marvin Minsky, "A Framework for Representing Knowledge," *Readings in Knowledge Representation*, eds. Ronald Brachman and Hector Levesque (Los Altos, Calif.: Kaufmann, 1985): 245–262.

6. R. Roberts and Ira Goldstein, "The FRL Primer," *MIT AI Lab Memo*, no. 408 (1977).

7. Harold Abelson and Gerald Sussman, *Structure and Interpretation of Computer Programs* (Cambridge, Mass.: MIT Press, 1985).

8. Eugene Charniak, Christopher Riesbeck, and Drew McDermott, *Artifical Intelligence Programming* (Hillsdale, N.J.: Lawrence Erlbaum, 1980).

9. Roger Schank and Christopher Riesbeck, *Inside Computer Understanding* (Hillsdale, N.J.: Lawrence Erlbaum, 1981).

10. James Allen, "Recognizing Intentions from Natural Language Utterances," *Computational Models of Discourse*, eds. Michael Brady and Robert Berwick (Cambridge, Mass.: MIT Press, 1983): 107–166.

DDJ

(Listings begin on page 58.)

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

Topics in Knowledge-Based Languages

Presenting a workable compromise between the rigid structures of Prolog, and the chaos of hypertext.

by Bill and Bev Thompson

If you look at the conference proceedings of the American Association for Artificial Intelligence, you'll see the papers broken down into categories such as AI architectures, cognitive modeling, knowledge representation, machine learning, and knowledge acquisition. The key words here are *architecture*, *modeling*, *representation*, and *machine learning*. The interest of much AI work has been to mimic human intelligence through the design of structures that let machines store and manipulate knowledge.

The users of AI systems, on the other hand, are interested in a totally different aspect of the problem. The lure of AI to corporations, scientists, educators, and individual authors is in the ability to communicate expertise via the medium of the computer. The current interest in hypertext systems represents another approach to the use of the computer to communicate complex knowledge. Because the ability to build structures is necessary in order to turn information and data into knowledge, any scheme that hopes to manipulate knowledge must be built around powerful but flexible structural concepts.

Bev and Bill Thompson are directors at Knowledge Garden Inc., Nassau N.Y. They have written extensively about AI systems and are the authors of KnowledgePro, KnowledgeMaker, and MicroExpert.

When we began our design project, which eventually became the language KnowledgePro, our goal was to build a system in which the focus was on the communication of expertise as opposed to its representation. This may sound like a meaningless distinction, but the switch in perspective affects the complexion of the system. In any design process, the designers are forced to make compromises. In saying that we would emphasize communication, we meant that when we made design decisions, we would compromise on the side of ease of use and expressive power rather than code efficiency.

In this article we will describe a structure called a *topic* that is the basic structural and storage unit used in KnowledgePro. A topic is a simple, uniform structure that provides both data representation and control. Topics serve many purposes; they can behave as procedures, functions, variables, system commands, and hypertext nodes. We'll discuss how topics support backward chaining, inheritance, and list processing as well.

Topics As Procedures

In its most basic incarnation, a topic is very much like a procedure in a language such as Pascal. It begins with a declaration of its name and parameters and ends with an *end* statement. Between the beginning and the end, a series of program statements are included. Example 1, page 41, shows an example of several nested topics.

Before discussing the topics in this example, we should say a few words about KnowledgePro syntax. KnowledgePro statements consist of the name of the command followed by a set of parentheses that enclose any parameters to be passed to the command. Each command ends with a period. Several of the most commonly used commands also have shorthand notations that make them more natural to express. Strings that include spaces or delimiting characters are enclosed within single quotes. Words included between (* and *) are comments.

KnowledgePro programs are called knowledge bases. All commands in the knowledge base are associated with an undeclared topic called *!main*. Topics nested within *!main* are executed in one of two ways: directly using a *do* command or through a mechanism called backward chaining. Using the *do* command is just like executing a procedure in C or Pascal. The command can be written using the uniform notation that uses the name of the command:

```
do ('which language').
```

or with a format more familiar to C and Pascal programmers:

```
'which language'( ).
```

In either case parameters can be passed to the topic much as in C or Pascal.

Backward Chaining

Backward chaining is a technique used in PROLOG and by rule-based expert system shells. In a "conventional" programming language, when you use a variable that hasn't yet been assigned a value, the value of the variable is unpredictable. In a language that uses topics, there are no variables as such. The topic serves not only as a structural unit but also as the unit of storage. Values can be both assigned to and retrieved from a topic. The dual nature of the topic makes it possible for an unassigned value to cause the execution of a topic in an effort to find its value.

As an example, suppose we change the first line of the knowledge base in Example 1 to read:

```
if ?'more about languages' is yes
   then do ('which language').
```

and we add the following topic:

```
topic 'more about languages'.
   ask ('Do you want to know more
about languages ?',
        'more about languages',
                              [Yes,No]).
end. (* more about languages *)
```

Here, the *?* is a shorthand notation for the *value_of* command, so this expression can also be written as *value_of ('more about languages')*. *value_of* requests a value for the topic *more about language* and, not finding a value, causes the commands associated with the topic *more about languages* to be executed. This topic contains an ask command that uses three parameters: the question, the name of the topic in which the answer is saved, and optionally a list that will place a menu of options on the screen. In this example the answer to the question is saved in a topic that already exists. If the topic selected was one that was not defined in the knowledge base, it would be created.

If a statement containing *?'more about languages'* is encountered again in the knowledge base, it will not trigger the execution of the *ask* command but will merely retrieve the value of the topic. Of course, a topic can be considerably more complex than the one shown in the example. A topic whose execution has been triggered by the *?* operator could trigger the execution of other topics either directly or through backward chaining.

Topics As Variables

The *?* operator retrieves the value from a topic. A topic may be assigned a value either as the result of executing commands nested within itself, or it may be assigned a value as the result of another topic's actions.

Along with values and associated procedures, topics have a series of properties that describe the number and kinds of values they may take on. By default, topics are unrestricted in the number and kinds of values that may be assigned to them. It is possible, however, to restrict topics to a range of numerical values, to a limited number of values,

```
topic 'which language'
  ask ('What #mlanguage#m do you want to know more about ?',want,
       [Pascal,C,Lisp,Prolog]).
  do (?want).

  topic language.
    window ().
    say('
         Some people make a distinction between AI and
         "conventional" programming languages.  Though individual
         languages certainly differ, for the most part, an AI language
         is one that was developed at a place where they happen to be
         doing AI.').
    close_window ().
  end. (* language *)

  topic Pascal.
    say ('
        When using Pascal to solve "AI types" of problems, you
        usually have to design low-level routines to implement linked
        list structures.').
  end. (* Pascal *)

  (* topics for C, Lisp and Prolog would go here *)

end. (* which language *)
```

Example 1: *Fragment of a KnowledgePro program, or knowledge base*

```
say ('One of the powerful features of #mLisp#m is the ability
to easily manipulate #mlist structures#m.').

topic mark (find).
  text = read ('threads.fil', concat ('/',?find), '/end').
  window ().
  say (?text).
  close_window ().
end. (* mark *)
```

Example 2: *Using the default topic* mark *with hypertext*

```
topic animal.
  :legs = 4. (* The default number of legs for an animal *)
  dog ().    (* These commands are used for initialization *)
  cat ().
  bird ().
  say ('
       A dog has ',?dog:legs,' legs.
       A cat has ',?cat:legs,' legs.
       A bird has ',?bird:legs,' legs.').

  topic dog.
  end. (* dog *)

  topic cat.
  end. (* cat *)

  topic bird.
    :legs = 2. (* override the default *)
  end. (* bird *)

end. (* animal *)
```

Example 3: *Nested topics showing how values are inherited*

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

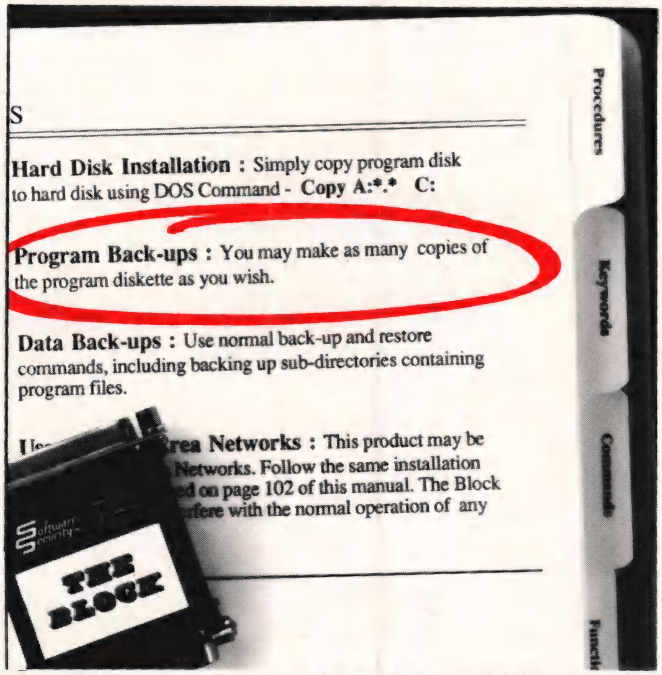
Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905

203 329 8870

CIRCLE NO. 127 ON READER SERVICE CARD

or to members of a set of specified legal values.

Topics may be predefined in the knowledge base or created on the fly. This has the advantage of allowing the creation of complex structures at run time. For example, the statements:

```
ask('Which cat do you wish to choose
?',cat_name, [Figaro,Flo, Babe]).

?cat_name = 'needs a new flea
                          collar.'
```

will produce a menu of choices and allow the user to choose one or several names. For each name chosen, a topic will be created if it doesn't exist and the topic will be assigned a string.

Designers of structured languages tell us that all data structures should be predefined and strongly typed. A reason often cited for this is that programs designed in this way are easier to debug and maintain. Actually, debugging and maintenance are functions of the programming environment. Strong typing and predefinition of data structures make the design of efficient compilers easier because at run time the program contains the location and type of most of its data structures. Languages that create structures at run time are dependent on efficient search routines to find structures.

Rules

In the jargon of expert systems, an *if. . .then* statement is called a rule. Rules can be expressed in the form shown earlier but can also share the formal notation of all KnowledgePro commands. The rule in our example has the following internal representation:

```
rule(eq(?('more about languages'),
yes), delay([do ('which language')])).
```

delay is used to prevent the second half of the rule from evaluating until the first part is evaluated and returns a Boolean value of *T*.

As you might suspect from looking at the example, system com-

We'll Give You Six Solid Reasons Why You Should Be Developing Your Applications In Clarion.

All New Version 2.0

The Clarion Professional Developer Is A Total Programming Environment That Runs On Any IBM PC, PS/2, Or True Compatible With 380K Of Memory And A Hard Disk. The Retail Price Is Just $695. NOT Copy Protected.

CLARION
PROFESSIONAL DEVELOPER™

CIRCLE NO. 130 ON READER SERVICE CARD

1 Slash your Total Development Effort... From Prototyping To Completion

You'll get live results fast—even before you write the first line of code. Ideas become running applications in a few hours.

Designer, the front-end application generator, allows you to produce major programs with *no coding*. It eliminates prototyping as a preliminary step because design and source code generation are done concurrently. You can implement a dazzling color screen or a report in minutes. You'll probably never code a screen again!

2 Automatically Generate Commented Source Code For Your Entire Application

The Professional Developer's **Designer** is the most powerful application generator in the industry. It creates structured source code that is fully commented. You can easily add to it, modify it, or just admire it.

And The Professional Developer is a complete development environment with all the tools that you need.

3 Create High-Speed, Bullet-Proof Data Management With Built-In LAN Support

Advanced techniques allow you to tune your data management to fit each application. Use related files, data encryption, memo fields, automatic recovery, and commit and rollback—all without compromising Clarion's high level of performance. And complete LAN features are included so the applications you develop will run on your network without any additional run-time or LAN PACK cost.

4 Interface To Your Own C or Assembler Routines For Special Requirements

The Professional Developer will support device drivers and complex logic written in C and Assembler. You won't have to re-write all those existing routines. You can produce completely open-ended solutions that can grow as your needs or requirements change.

5 Produce Executable Programs That Don't Need Run-Time Systems

Compile your application into an .EXE program that will run on a stand-alone computer or workstation so you can distribute your programs without costly run-time systems.

6 Getting Started Is Easy. You'll Be Productive The First Day.

Although there's a lot of horsepower "under the hood," you'll be producing programs soon after you open the package. You'll find the whole environment friendly and comfortable. PC Week says: "Clarion is easy to learn and easy to use."

And One More.

We'll Give You A Free Preview!

See Your Dealer Or Call Toll Free

(800) 354-5444

For A Free Copy Of Our Tutorial Diskette And Introductory Material

(Or, simply return this coupon).

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone (_____) _____

Mail This Coupon To:

Clarion Software
150 East Sample Road, Pompano Beach, FL 33064

mands are themselves topics defined internally but behave in all other ways like user-defined topics. In an application the internal topics can be overridden by defining a topic with the same name within the knowledge base or by using a topic written in another language and included in an external library. The same technique can be used to create new topics that extend the language.

Hypertext and Topics

Returning to the example in Example 1, let's look at the first line in the topic *which language*:

```
ask ('Which #mlanguage#m would
you like to know more about ?',
    want, [Pascal,C,Lisp,Prolog]).
```

We've already described how the *ask* command works, but if you look closely, you'll notice that the text of the question includes the word *language* enclosed within the characters *#m*. This notation is used to define a hypertext node. Any text marked in this way is displayed on the screen in inverse video (or in any color selected by the application designer). The user of the application can use a function key or a mouse to move among highlighted concepts. When a concept is selected, the topic of the same name—for example, *language*—is executed. The commands, *window*, *close_window*, and *say* are self-explanatory. Here, *window* and *close_window* use default values, but they can also include parameters that specify title, color, size, and location of the windows selected.

If no topic with the same name can be found, the system will next search for a topic called *mark* and, if it finds it, will pass the hypertext phrase to *mark* as a parameter. Example 2, page 41, shows an example of one way in which this feature is used. In the knowledge base in Example 2, all the text threaded to hypertext nodes is stored in an external data file called *thread.fil*. When the *say* command displays its message, the *Lisp* and *list structures* are highlighted, but no topics with

The Janus/Ada Difference... More Than Just the Price!

Contrary to what you may believe, all Ada compilers are not equal. The popular misconception is that all Ada compilers are the same for any machine, just differently priced. But the price is just one of the differences. Compilation speed, the ability to emulate floating point, usability on all of the Intel 80 X 86 family, as well as support of networking sites, can make a big difference to your project. That big difference is the Janus/Ada difference!!!

Janus/Ada is substantially different from other Ada compilers. It was developed on the microcomputer, for the microcomputer, and was bootstrapped version by version. The resulting compiler is faster, more robust and more flexible than other Ada compilers. Minimizing the expenses of add-on hardware, run-time library fees and tutorials also makes Janus/Ada different. Take a look at the charts below and see what we mean when we say we're different.

Product Features:	Janus/Ada 2.0	Compiler A 3.2*	Compiler M 2.0*
80 X 87 emulation	YES	NO	NO
Royalty free run time libraries	YES	NO	NO
Site licensing	YES	NO	NO
All 80 X 86 covered	YES	NO	NO
Tutorial included with all Paks	YES	NO	NO
Ada applications for only $12	YES	NO	NO
Runs on floppy disks	YES**	NO	NO
Validation Suite	ACVC 1.9	ACVC 1.8	ACVC 1.8
Validated compiler cost	$99.00	$3,000.00	$795.00

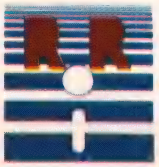
*Comparisons made on product information obtained on 12/11/87.
**3½", 720K Floppies and 5¼", 1.2M Floppies only.

The differences don't end with the facts above; the performance issues, which make or break a production compiler, demonstrate why Janus/Ada is not just different, but better!

Compile and Link Timings:**	Janus/Ada	Compiler A	Compiler M
Sieve	0:50	1:39	1:25
Calculation	0:43	2:14	1:00
Disk Write	0:45	1:33	1:25
Disk Read	0:43	1:34	1:23
Integer Sort	0:47	2:13	1:34
Dynamic Allocation	0:47	2:02	1:32
Matrix Inversion	0:46	2:24	1:39
Recursion	0:47	2:10	1:34

**(These results are from BYTE Magazine, July 1987 issue; full details on the tests, as well as the standard equipment used, can be found in that issue.)

Our seven years of providing quality Ada software to over ten thousand users reflects the commitment we have to your programming needs. Our policy has always put you, the customer, first and foremost. We think it's a difference you can appreciate! You can order our **"JET SET" (Ada compiler and tutorial) for only $99.** Please call our toll free number **1-800-722-3248 (1-800-PC ADA 4 U)** to place your order or request our products brochure.



SOFTWARE, INC.

specialists in state of the art programming

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168 1-800-722-3248

(continued from page 45)

these names are defined. If the user selects one of these—for example, *Lisp*—the topic mark is executed and passed *Lisp* as a parameter. The first line in *mark* assigns a value to the topic *text*, which is also created at this time. The value assigned is the result of the *read* command.

read uses three parameters: the name of the file to read, where to begin reading, and where to stop reading. The command *concat* returns */Lisp*, which is the result of concatenating the character / onto the value of the topic *find*. All text following this string and up to the string */end* is assigned to the topic *text*. A window is opened; the value of *text*, which might be several pages long, is displayed; the window is closed; and control returns to the statement from which the hypertext was selected.

Nested Topics and Inheritance

Topics can be nested inside other topics to form a hierarchy, which allows you to package pieces of a knowledge base into self-contained units. A topic defined within another topic is normally accessed only from within that topic. Normally, KnowledgePro searches for a topic by examining the topics defined within the current topic. If none of these is the one being sought, it examines the parent topic to see if the topic is defined there. KnowledgePro doesn't look at topics nested within those defined at the parent level. Nested topics are normally invisible from the outside; they are only found through the hierarchical search process. The search continues up the hierarchy until the topic is found or the topic *!main* is examined.

Example 3, page 41, shows an example of a nested topic. If *dog* is referenced outside of *animal*, it won't be found unless the full name *animal:dog*, which defines its location in the hierarchy, is specified. The hierarchical structure allows topics to inherit values from other topics. Example 3 shows a knowledge base that describes some properties of some common animals. The command *animal ().* will display the following message:

```
A dog has 4 legs.
A cat has 4 legs.
A bird has 2 legs.
```

The default value for *legs* is set to 4. The topics *dog*, *cat*, and *bird* are then executed. Of these topics only *bird* contains an override value. The *:* before the name *legs* in this topic tells KnowledgePro that this *legs* is a local topic, not the one defined in *animal*. Because this topic doesn't exist, it is created. When a value is sought for each of these items in the *say* statement, both *dog:legs* and *cat:legs* are assigned the default value of 4 because no override was specified.

List Processing

Lists are integral to all topics. Each parameter passed to a topic is actually a list. For example, the first parameter of an *ask* command is the question. So far, you've seen questions made of simple text expressions; however, because each parameter can be a list, more complex displays can be created, as in this example:

```
ask (['The total cost is $',
?price * ?'number of items',
   'How will you pay ?'],
            pay,[Check,'Credit card']).
```

Here, the question is a list containing three items: a literal string, the result of multiplying the value of the topic *price* by the value of the topic *number of items*, and a final literal string.

Turning back for a moment to Example 1, you can see another example of the use of lists. The response to the question *What language do you want to know more about?* can include more than one item from the list of menu items *[Pascal,C,Lisp,Prolog]*. The answers selected are stored as a list that is assigned to the topic *want*. Let's say that the answers selected were *C* and *Prolog*. When the command *do (?want).* is executed, each topic in the list assigned to *want* will be executed, so the first topic, *C*, will be performed and then the topic *Prolog* will be executed.

Concepts For Creating Your Own Solutions

Video Courses.

Artificial Intelligence in Depth

by Dr. Paul R. Cohen

▶ *The 25 lectures in this course provide a fast-paced introduction to Artificial Intelligence ideas, including search, planning and problem solving, expert systems, natural language, computer vision, logic, reason maintenance, reasoning under uncertainty, blackboards and sophisticated control, and machine learning. The course is taught as part of the graduate curriculum at the University of Massachusetts, and features guest lectures by Professors Victor Lesser, Allen Hanson, and Daniel Corkill. The lectures and review questions focus on research issues, though no prior background in AI is required of the diligent student.*

Dr. Paul Cohen is an Assistant Professor of Computer Science at the University of Massachusetts, Amherst, where he is Director of the Experimental Knowledge Systems Laboratory. He is a co-editor of *The Handbook of Artificial Intelligence.*

Expert Systems: Theory and Practice

by Dr. Paul R. Cohen

● *This highly popular course is now available as a self-study videotape training package, including 12 hours of videotaped instruction, detailed lectures notes, over 300 illustrations, study questions, a bibliography, and suggestions for further reading. The lectures consist of an introduction, discussions of rule-based, frame-based, and optimization-hybrid systems. The second half of the course addresses research issues including control, uncertainty, learning, explanation and knowledge engineering.*

Dr. Daniel Corkill is a Research Computer Scientist at the University of Massachusetts, Amherst.

Programming in Common Lisp

by Dr. Daniel D. Corkill

■ *This series of eight, one and one-half hour sessions on Programming in Common Lisp was developed for programmers with no experience with Lisp. Topics include recursion and mapping, functions, complex data structures, control forms, I/O, macros, and compiling. The course emphasizes programming style. Each videotape contains several exercises to be attempted while the tape is paused; each exercise is subsequently discussed with detailed examples. Ideally, two or more individuals will work together on a Lisp system to discuss and solve the exercises.*

He is the originator of CLISP, has extended Common Lisp for multiprocessing, and recently developed a generic blackboard implementation system called GBB.

The Minerva Group is a team of researchers and teaching professionals that provides advanced education on AI technology currently being developed at research centers throughout the United States. The courses present the knowledge required to build state-of-the-art systems as well as concepts and techniques for solving today's expert systems problems.

The Minerva Group

Call **(603) 643-6349** or Write to P.O. Box 835, Amherst MA 01004.

Lists enable you to manipulate groups of related items. In one knowledge base we examined recently, the designer had a long set of rules with each rule having only one conditional clause, as in this example:

```
if ?lot is K10
   then item is 3A62.

if ?lot is A22
   then item is 6H77.
...
topic lot.
ask ('What is the lot number on the
remaining section ?',lot,
   [K10,A22,...]).
end.
```

This entire set of rules, no matter what its length, could have been expressed in four lines using lists:

```
lot_list is [K10,A22,...].
item_list is [3A62,6H77,...].
ask ('What is the lot number ?',
                        lot,?lot_list).
item is element (?item_list,
   where (?lot_list, ?lot) ).
```

When the question is displayed, the contents of the topic *lot_list* are displayed on the menu. The item selected—for example, *6H77*—is assigned to the topic *lot*. The evaluation of the last line begins by finding where the value selected, *6H77*, is located in the list *lot_list*. The value 2 is returned as the result of the *where* command. Next, the *element* command returns the second item of the list contained in *item_list* and this value is assigned to the topic *item*.

Many other list-handling commands are necessary for a high-level symbolic language. These include commands such as *first*, *last*, and *rest* (similar to the LISP commands *car*, *last*, and *cdr*) that let you tear apart lists. KnowledgePro also includes commands for combining, comparing, and constructing lists in a variety of ways.

Summary

The flexibility of the hypertext and the directness that comes with procedural control make it possible for people with little or no programming experience to sit down and easily express their expertise. At the same time, techniques such as backward chaining, inheritance, and list handling enable it to be used in complex symbolic computational problems. With languages built around these types of structures, it becomes possible to design applications that allow program control to be shared by both the designer and the user of the system, surpassing the rigidity of the current expert system technology and the free-form environment of hypertext media.

Readers interested in seeing a sample of the KnowledgePro environment can obtain a run-time version from CompuServe: go PCVEN (data library 8). Several free demo programs, including TextPro (a program for writing and reading short hypertext documents), are available. Source code is included with the demos so you can get an idea of how topics work.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

Theorem Proving Using Semantic Resolution

Resolving those nasty semantic clashes with C

by Anthony J. Dos Reis

One expects the *resolution principle*—a simple rule of inference useful in theorem proving—to date back to antiquity, along with *modus ponens* and Euclid's *elements*. In fact, though, it was discovered as recently as 1965 by J.A. Robinson and constituted a major breakthrough in mechanical theorem proving. It is surprising that such a powerful and (it seems now) obvious result did not surface much earlier.

Since 1965, the resolution principle has undergone many refinements. One such refinement—semantic resolution—is discussed in this article. To avoid becoming mired in the complexities of formal logic, I will restrict my attention to propositional logic. Although the power of propositional logic is limited, it is nevertheless an ideal vehicle for presenting the essential aspects of semantic resolution. (See Listing One page 64.)

Basic Definitions

I will use the symbols &, v, –>, and ¬ to represent the logical operations conjunction, disjunction, implication, and negation, respectively.

An expression (called the *conclusion*) is said to *logically follow* from a set of expressions (called the *premises*) if the conclusion is true whenever the premises are all true. A conclusion with its premises is called an *argument*. An argument is *valid* if its conclusion logically follows from its premises. The conclusion in a valid argument is called a *theorem*.

A *literal* is a variable or the negation of a variable. A *clause* is either a single literal or a disjunction of literals or is *empty* (denoted by the symbol [ ]). Examples of clauses are [ ], p, $\neg q$, $p \vee \neg q$, and $\neg p \vee q \vee r$. A variable and its negation (for example, p, $\neg p$) is called a *complementary pair*. The assignment of a truth value to every variable under consideration is called an *interpretation*. A set of clauses is *unsatisfiable* if for every interpretation at least one clause in the set is false.

Expressions with identical truth table values are said to be *equivalent*. It can be shown that any logical expression can be transformed to an equivalent expression consisting of either a single clause or a conjunction of clauses. For example, the expression $(p \vee q) \rightarrow r$ can be converted to the equivalent expression $(\neg p \vee r) \,\&\, (\neg q \vee r)$. Thus, without loss of generality, we can restrict our attention to logical expressions of this standard form. In fact, we can restrict our attention to individual clauses because a conjunction can be viewed as several individual premises or conclusions. For example, the premise $(\neg p \vee r) \,\&\, (\neg q \vee r)$ can be viewed as two premises—$(\neg p \vee r)$ and $(\neg q \vee r)$—each a single clause.

To prove that a conclusion logically follows from its premises in propositional logic, you simply need to construct a truth table and determine if the conclusion meets the definition of logically follows. Although this approach is eminently satisfactory for propositional logic, it unfortunately falls apart in more powerful logic systems (such as first-order logic) in which truth tables may not be of finite size. For such systems rules of inference are used to establish whether a conclusion logically follows from its premises. A *rule of inference* is a rule that describes how to generate a new expression that logically follows from existing expressions. The step-by-step derivation of the conclusion from the premises using rules of inference is called a *proof*.

Typically, you need many rules of inference to generate the expressions needed in a proof. For example, the rule of inference *modus ponens* given by:

```
E1 –> E2
E1
∴ E2
```

Anthony J. Dos Reis is an assistant professor at the College at New Paltz, Dept. of Mathematics and Computer Science, New Paltz, NY 12561.

cannot be used to show that $\neg p$ logically follows from $p \rightarrow q$ and $\neg q$ because the premises are not of the appropriate form. *Modus ponens* by itself is not *complete*—that is, there are valid arguments whose conclusions cannot be generated using *modus ponens* alone.

A serious problem with having many rules of inference is that it is necessary to select the appropriate rule at each step in a proof. The appropriate rule is, in general, not obvious. You could, of course, try every rule at every step, but such a brute-force approach is very inefficient in time and space.

The Resolution Principle

The resolution principle is a rule of inference given by the four forms shown in Table 1, below.

Resolution requires two clauses (the clauses to be resolved) with a complementary pair—that is, one clause should contain some expression *E1* and the other $\neg E1$. The *resolvent*—that is, the clause generated—is formed from the expressions that remain after *E1* and $\neg E1$ are thrown away. If more than one expression remains (as in the first form in Table 1), the resolvent is the disjunction of these expressions. If no expressions remain (as in the fourth form in Table 1), the resolvent is the empty clause, denoted by *[ ]*. Because disjunction is commutative, the order of the expressions is not important.

If two clauses that both contain some expression E are resolved, the resolvent need not have two occurrences of E because E v E always equals E. For example, you can take the resolvent of p v q v r and $\neg$p v q to be q v r rather than q v q v r.

If clauses that have more than one complementary pair are resolved, only one complementary pair is eliminated. For example, if you resolve $p \vee q$ and $\neg p \vee \neg q$, you get either $q \vee \neg q$ or $p \vee \neg p$, depending on which complementary pair you eliminate. Here the resolvent is always a tautology (that is, an expression that is always true) and therefore logically follows from any set of premises. In refutation proofs (see later), such resolutions are useless and should never be performed.

In the following example, 6 through 9 are obtained by resolution. Each is labeled with the line numbers of the expressions from which it is derived.

```
1 p v ¬q v r
2 p v q v s
3 ¬p
4 ¬r
5 ¬s
6 p v r v s      (1,2)
7 p v r          (5,6)
8 r              (3,7)
9 [ ]            (4,8)
```

The rule of inference *modus ponens* given in clause form:

```
¬E1 v E2
  E1
∴ E2
```

is clearly a special case of the resolution principle. Other rules of inference—*modus tollens*, *hypothetical syllogism*, *disjunctive syllogism*, *constructive dilemma*, and *destructive dilemma*—are also special cases of resolution. That resolution encompasses all these rules is an indication of its power and generality.

Proof by Refutation

Suppose you wish to show that the argument:

```
P1
 .
 .      premises
 .
Pn
∴ C     conclusion
```

is valid—that is, *C* logically follows from *P1*, *P2*, . . ., *Pn*. The straightforward approach is to derive C from the premises using rules of inference. Another approach—called proof by refutation—is to negate the conclusion and then derive the empty clause from the premises and the negated conclusion. I'll first look at an example and then discuss why this approach works.

To show that the following argument is valid:

```
¬p v r
¬r v t      premises
  p
∴ T         conclusion
```

first negate the conclusion and add it to the premises. Then using rules of inference, derive the empty clause:

```
1 ¬p v r
2 ¬r v t
3 p
4 ¬t      negation of the conclusion
5 r       (1,3)
6 ¬r      (2,4)
```

(i)	(ii)	(iii)	(iv)
E1 v E2	E1 v E2	E1	E1
¬E1 v E3	¬E1	¬E1 v E3	¬E1
∴E2 v E3	∴E2	∴E3	∴[]

Table 1: *The four forms of resolution*

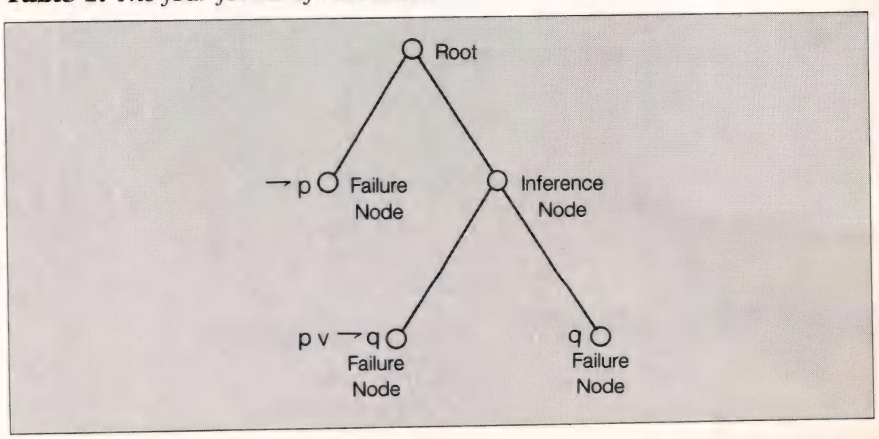


Figure 1: *Binary tree*

(continued from page 51)

7 [ ] (5,6)

To understand why this approach works, first note that the generation of the empty clause (line 7) requires the resolution of an expression and its negation, *r* and $\neg r$. Now assume that the initial clauses (1, 2, 3 and 4) are all true. Then both *r* and $\neg r$ must also be true because they are derived from the initial clauses using a rule of inference. But because *r* and $\neg r$ always have opposite truth values, they cannot both be true. The assumption—that the initial clauses are all true—must be false. Thus, whenever clauses 1, 2, and 3 are all true, clause 4 (the negation of the conclusion) is false and the conclusion is true.

Although proof by refutation seems like a roundabout way of proving a theorem, it is actually more suitable for machine implementation than the straightforward approach for two reasons. First, the goal of the refutation approach is an empty clause, which is generally easier to derive because it may be obtained from any one of perhaps several complementary pairs. For example, another possibility for the previous proof is to derive $\neg t$ and *t*, as follows:

1 $\neg$p v r
2 $\neg$r v t
3 p
4 $\neg$t negation of the conclusion
5 r (1,3)
6 t (2,5)
7 [ ] (4,6)

Second and even more important, resolution is complete when used in a refutation proof—that is, if the original argument is valid, then the empty clause can always be derived using only the resolution principle. Our bagful of inference rules is not needed.

Completeness of Resolution

The basic idea in the proof of the completeness of the resolution principle can be illustrated by considering a simple example. Suppose the set of clauses from which the empty clause is to be derived is $\{\neg p, p \vee \neg q, q\}$. Construct a binary tree as follows:

1. Start with an unlabeled root node.

Now repeat steps 2 and 3 for each variable *V* that appears among the set of clauses (in this example, steps 2 and 3 will be done twice—once for $V = p$ and once for $V = q$):

2. Extend the tree from any unlabeled leaf node by adding two branches that terminate on two new nodes, the left and right of which represent the assignment of true and false, respectively, to *V*.
3. If the truth assignment associated with all the nodes in the path from the root to a node added in step 2 makes a clause false, then label that node with that clause. (Labeled nodes are called failure nodes.)

The tree constructed corresponding to this example is shown in

I N T R O D U C I N G

The Graphics Toolkit for Contemporary Software Developers

Already the fastest and most powerful graphics toolkit on the market, the new HALO® delivers subroutines and device support for exciting, contemporary applications in publishing, office automation, vision, and image processing.

HALO '88 is a device independent library of 190 graphics subroutines. It is compatible with 18 programming languages, and over 140 hardware devices such as image scanners; graphics, vision, and imaging boards; printers and plotters; and mice. HALO '88 is designed for the complete IBM compatible microcomputer line including the PS/2 and VGA.

Today's Tools for Tomorrow's Applications

HALO '88 has new subroutines which control scanners and scanned images — even images which are larger than screen resolution and available memory. Extended character set support enables software developers to address IBM's full 255 characters in graphics and to design foreign language fonts. Among contemporary HALO '88 applications are CAD, Computer-Based Training, Presentation Graphics, Graphic Arts, Mapping, Machine Vision, Silicon Wafer Manufacturing, Sound System Design, Vehicle Scheduling and Routing, and Real Estate.

Join the HALO Family

HALO has an installed base of 60,000+ end-users, hundreds of site-licensed corporations, government agencies, universities, and national laboratories and most importantly, over 220 Independent Software Developers (ISVs) who market applications written with HALO.

HALO '88 provides the software designer with the richest environment of graphics functions; the programmer with reliable and well-documented tools, and DP managers with continuity of user interface and database format.

Reach for the Future

If you need high performance graphics development software that provides a migration path to OS/2 and other future technology, follow the industry leaders — call (800) 992-HALO (4256).

HALO '88 is just $325 and includes all device drivers, 20 fonts, your choice of one compiler binding, completely new documentation, an interactive tutorial and free 800# technical support. Update from HALO for $150.

Ask about the new HALO Programmers' Workbook which provides C program examples for HALO '88 applications developers.

HALO™
infinite possibilities in graphics and imaging

media cybernetics
8484 Georgia Ave.
Silver Spring, MD 20910
(301) 495-3305, (800) 992-HALO

(continued from page 52)

Figure 1, page 51. Observe that every leaf node must be a failure node because otherwise an interpretation would exist that satisfies the set of clauses. Moreover, at least one node (called an inference node) must have two failure nodes immediately below it. Because the truth assignments associated with these failure nodes differ for exactly one variable, their corresponding clauses must contain exactly one complementary pair and are therefore resolvable. Furthermore, the resolvent is false (in the sense of step 3 above) for the inference node. Thus, the inference node can be labeled with the resolvent (making it a failure node) and the two nodes below it pruned. Thus, you get the tree shown in Figure 2, page 55.

For resolution to be practical, it must be restricted from generating too many unnecessary resolvents

The same argument can be applied repeatedly, resulting in further pruning (and resolution) until only the root node remains, corresponding to the derivation of the empty clause.

Semantic Resolution

In a resolution proof, the clauses to be resolved must be selected at each step. In general, there may be many resolvable clauses that produce resolvents that are not required for the proof. For example, consider:

1 p v q
2 p v ¬q
3 ¬p v q
4 ¬p v ¬q
5 p (1,2)

6 →p (3,4)
7 [ ] (5,6)

The resolvents *(1,3)*, *(2,4)*, *(1,6)*, *(2,6)*, *(3,5)*, and *(4,5)* are not needed for the proof and are, therefore, wasteful to generate. A mechanical theorem prover, however, unless otherwise constrained, would typically generate such resolvents. For resolution to be practical, it must be restricted from generating too many unnecessary resolvents. The restrictions used, however, must still allow the necessary resolvents to be generated, or else the technique would not be complete.

One approach to restricting resolution that is complete is called *semantic resolution*. Semantic resolution applies two restrictions to resolution: one using an interpretation; the other, an ordering. Recall that an *interpretation* is the assignment of a truth value to each variable. In the following set of clauses, for example:

1 p v q
2 p v →q
3 →p v q
4 →p v →q

p can be assigned true and *q* false. With this particular interpretation, clauses 1, 2, and 4 all evaluate to true, and clause 3 evaluates to false. For a given interpretation, a clause that evaluates to true is called a *nucleus* (denoted by a +), and a clause that evaluates to false is called an *electron* (denoted by a –). An *ordering* is simply a listing, in descending priority, of the variables in a set of clauses. A single interpretation and a single ordering should be used for the entire proof. The particular interpretation and ordering used is not critical, however—any will work.

The two restrictions of semantic resolution are:

1. Never resolve a nucleus with a nucleus.
2. Resolve an electron with a nucleus only if the variable to be eliminated has the highest priority among the variables that appear in the electron.

An example should make these two rules clear. For an interpretation, assign true to *p* and *q* and take <*p*, *q*> as the ordering. Now consider the following clauses:

1 p v q (+)
2 p v →q (+)
3 →p v q (+)
4 →p v →q (–)

The possible resolvents are *(1,2)*, *(1,3)*, *(2,4)*, and *(3,4)*. By restriction 1, *(1,2)* and *(1,3)* should not be generated because both are nuclei. By restriction 2, *(3,4)* should not be generated because the variable *p* appears in the electron and has higher priority than *q*, the variable eliminated. Thus, the only allowable resolvent is *(2,4)*:

5 →q (2,4), (–)

With clause 5 added to the list, the only possibilities now are *(1,5)* and *(3,5)*:

6 p (1,5), (+)
7 →p (3,5), (–)

Now the possibilities are *(4,6)*, *(1,7)*, *(2,7)*, and *(6,7)*:

8 →q (4,6), (–)
9 q (1,7), (+)
10 →q (2,7), (–)
11 [ ] (6,7)

Two additional points should be

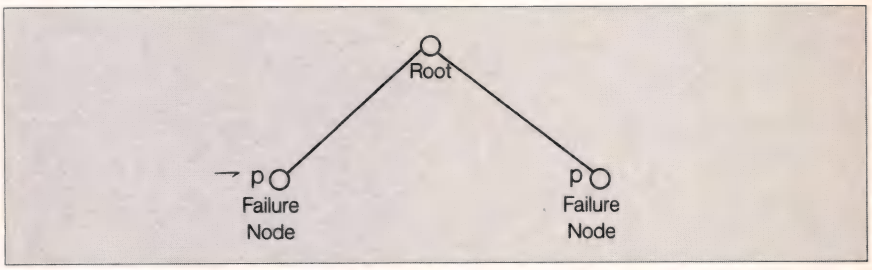


Figure 2: *Binary tree after resolution*

10 Important Reasons C Programmers Use Our File Manager

1. It's written in C.
Clearly the growing language of choice for applications that are fast, portable and efficient. All of db_VISTA's source code is written in C.

2. It's fast – almost 3 times faster than a leading competitor.
Fast access that comes from the unique combination of the B-tree indexing method and the "network" or direct "set" relationships between records. A winning combination for fast performance.

3. It's flexible.
Because of db_VISTA's combination of access methods, you can program to your application needs with ultimate design flexibility. Use db_VISTA as an ISAM file manager or to design database applications. You decide how to optimize run-time performance. No other tool gives you this flexibility without sacrificing performance.
db_VISTA is also well behaved to work with most any other C libraries!

4. It's portable.
db_VISTA operates on most popular computers and operating systems like UNIX, MS-DOS and VMS. You can write applications for micros, minis, or even mainframes.

5. Complete Source Code available.
We make our entire C Source Code available so you can optimize performance or port to new environments yourself.

6. It uses space efficiently.
db_VISTA lets you precisely define relationships to minimize redundant data. It is non-RAM resident; only those functions necessary for operation become part of the run-time program.

7. Royalty free run-time.
Whether you're developing applications for yourself or for thousands, you pay for db_VISTA or db_QUERY only once. If you currently pay royalties to someone else for your hard work, isn't it time you switched to royalty-free db_VISTA?

db_VISTA™

Features
- **Multi-user** support allows flexibility to run on local area networks
- **File structure** is based on the B-tree indexing method
- **Transaction processing** assures multi-user consistency
- **File locking** support provides read and write locks
- **SQL-based db_QUERY** is linkable
- **File transfer** utilities included for ASCII, dBASE optional
- **Royalty-free** run-time distribution
- **Source Code** available
- **Data Definition Language** for specifying the content and organization of your files
- **Interactive database access** utility
- **Database consistency check** utility

File Management Record and File Sizes
- Maximum record length limited only by accessible RAM
- Maximum records per file is 16,777,215
- Maximum file size limited only by available disk storage
- Maximum of 256 index and data files
- Key length maximum 246 bytes
- No limit on number of key fields per record
- No limit on maximum number of fields per record

Operating System & Compiler Support
- **Operating systems:** MS-DOS, UNIX, XENIX, Macintosh.
- **C compliers:** Lattice, Microsoft, IBM, Turbo C, XENIX, UNIX and Light SpeedC.

8. SQL-based db_QUERY™
Add our new C-linkable, SQL-based, ad hoc query and report-writing companion product to provide a simple relational view of your db_VISTA applications. Without compromising speed.

9. Free tech support.
60 days of free technical and application development support for every Raima product. Of course, extended support and training classes are also available at your place or ours.

10. Upward database compatibility
Start out with file management in a single-user PC environment—then move up to a multi-user LAN or a VAX database application with millions of records. You'll still be using db_VISTA. That's why so many C programmers are choosing db_VISTA.

But don't just take our word for it.
"db_VISTA has proved to be an all-round high performer in terms of fast execution, flexibility and portability, and has undoubtedly saved us much time and development effort."
John Adelus, Hewlett-Packard Office Productivity Division

30-day Money Back Guarantee!
Try db_VISTA in your environment for 30 days and prove it to yourself. If not completely satisfied, return it for a full refund.

Call The Programmer's Shop
Toll Free Today For Our Special Offer
1 (800) 421-8006
[In MA 1 (800) 442-8070]

THE PROGRAMMER'S SHOP
Your complete source for software, services and answers

5 Pond Park Rd.
Hingham, MA 02043
(617) 740-2510

RAIMA™ CORPORATION

CIRCLE NO. 141 ON READER SERVICE CARD

SEMANTIC RESOLUTIONS
(continued from page 55)

made about semantic resolution. First, if an interpretation makes the clauses all true or all false, then the original argument is not valid and, therefore, cannot be proved. Second, an electron can never be resolved with an electron because resolution requires a pair of clauses with complementary expressions, one of which must be true regardless of the interpretation used. Thus, one of the clauses is always a nucleus.

Semantic Clashes

Consider the following proof using semantic resolution with true assigned to *p*, *q*, and *r* as the interpretation and with <*p*, *q*, *r*> as the ordering. All the resolvents that can be generated under semantic resolution are listed.

```
1 p v ¬q v r          (+)
2 q                   (+)
3 ¬p v ¬q             (–)
4 ¬r                  (–)
5 ¬q v r        (1,3) (+)
6 p v ¬q        (1,4) (+)
7 ¬q            (4,5) (–)
8 ¬q            (3,6) (–)
9 [ ]           (2,7)
```

The true literals, *p* and *r*, in the nucleus on line 1 can be eliminated, respectively, by the electron on line 3 and the electron on line 4. Clauses 1, 3, and 4 constitute a *semantic clash*—that is, a single nucleus together with a set of electrons that eliminate all its true literals under semantic resolution. A semantic clash contains exactly one electron for each true literal in its nucleus.

The nucleus and electrons in a semantic clash can be resolved in the following manner. First, resolve the nucleus with any electron in the semantic clash. Then resolve the resulting resolvent with another electron. Continue resolving electrons with successive resolvents until all the electrons are used. For example, for the semantic clash *(1,3,4)* above, you get:

```
1 p v ¬q v r          (+)
2 q                   (+)
3 ¬p v ¬q             (–)
4 ¬r                  (–)
5 ¬q v r        (1,3) (+)
intermediate resolvent
6 ¬q            (4,5) (–)
final resolvent
```

The final resolvent of a semantic clash is always an electron or the empty clause—never a nucleus.

The importance of resolving semantic clashes is that the intermediate resolvents are never needed to produce the empty clause and therefore can be omitted. Fewer unnecessary resolvents are generated, resulting in a more efficient proof. The previous proof with nine lines, redone with semantic clashes, becomes a proof with only six lines:

```
1 p v ¬q v r (+)
2 q (+)
3 ¬p v ¬q (–)
4 ¬r (-)
5 ¬q (1,3,4) (–) semantic clash
6 [ ] (2,5) semantic clash
```

A semantic clash does not necessarily exist for every nucleus. For example, a semantic clash does not exist for clause 2 above until clause 5 is generated. However, because resolution restricted to semantic clashes is complete (for a proof, see Chang and Lee), there should always be at least one semantic clash until the empty clause is generated.

Bibliography

Chang, C.; and Lee, R.C. *Symbolic Logic and Mechanical Theorem Proving*. New York, N.Y.: Academic Press, 1973.

Winston, P.H. *Artificial Intelligence*. 2d ed. Englewood Cliffs, N.J.: Addison-Wesley, 1984.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 64.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

Listing One *(Text begins on page 18.)*

```
Listing One.  A simple adventure game written in AAL

(loc the-first-room
"You are in a small, gloomy room lit by an unseen source above you.
The walls and floor are smooth, hard and dark, like obsidian.  Exits
lead west and south."
  (contains whistle)
  (exits
   (w the-second-room)
   (s "You have wandered around and wound up back where you started")))


(loc the-second-room
"You are in a vast chamber of ice and rock.  Fiery torches in the walls
provide an eerie light.  There is a passageway south and another exit to
the north."
  (contains monster)
  (exits
   (s "The passageway is blocked by rubble.")
   (n (((alive monster) -> "The monster won't let you pass.")
       the-first-room))))

(command blow
        (blow *obj)
        (requires ((carrying player *obj) "You don't have a" *obj))
        "You can't blow that!")

(command (throw hurl chuck)
        (throw *instr at *obj)
        (requires (carrying player *instr)
                  (here *obj))
        "Nothing happens."))

(obj monster fixed
     (action throw *obj
             ("The monster destroys the a" *instr)
             (destroy *instr)))

(obj whistle
     (action blow *obj
             "The whistle emits a piercing screech."
             ((here monster) ->
              "The monster's eyes bug out--wider--wider--and then,
                finally, close forever."
              (dead monster))))
```

End Listing One

Listing Two

```
Listing 2 for Amsterdam's DDJ article: The deducer

(defun assert (list)
  (let ((new-stmt (list->rule list)))
    (if (and (add-to-database new-stmt) (eq (rule-type new-stmt) :fact))
        (run-forward-rules *assertion-rules* new-stmt)))
    new-stmt)

(defun retract (list)
  (let ((new-stmt (list->rule list)))
    (when (remove-from-database new-stmt)
      (run-forward-rules *retraction-rules* new-stmt))
    new-stmt))

(defun run-forward-rules (rules fact)
  ;; Run a rule if its antecedent matches the fact.  (Forward rules can
  ;; only have one antecedent.)
  (dolist (frule rules)
    (let ((bindings (unify fact (car (rule-antecedents frule)) *globals*)))
      (if (not (eq bindings :fail))
          (execute-action (rule-consequents frule) bindings)))))


(defun deduce (pattern-list bindings)
  ;; Returns a stream of bindings (variable lists) for things that match all
  ;; the patterns, or the empty stream if there are none.
  (if (null pattern-list)
      (stream-cons bindings *empty-stream*)
      (let ((bindings-stream (deduce-pattern (car pattern-list) bindings)))
        (stream-mapcan #'(lambda (b) (deduce (cdr pattern-list) b))
                       bindings-stream))))

(defun deduce-pattern (pattern bindings)
  (deduce-pat pattern bindings (find-possible-unifiers pattern)))

(defun deduce-pat (pattern bindings possibilities)
  (if (null possibilities)
      *empty-stream*
      (let* ((rule-pats (rename-rule (car possibilities)))
             (bindings1 (unify pattern (car rule-pats) bindings)))
        (if (eq bindings1 :fail)
            (deduce-pat pattern bindings (cdr possibilities))
```

(continued on page 60)

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492 (in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410D
So. Weymouth, MA 02190
(617) 337-6963

Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.

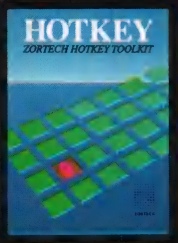
CIRCLE NO. 143 ON READER SERVICE CARD

Listing Two *(Listing continued, text begins on page 18.)*

```
            (stream-append
              (deduce (cdr rule-pats) bindings1)
              (deduce-pat pattern bindings (cdr possibilities))))))

(defun unify (pat1 pat2 bindings)
  ;;Returns :FAIL if it can't unify, a list of bindings if it can.
  (cond
    ((and (null pat1) (null pat2))
     bindings)
    ((or (null pat1) (null pat2))
     :fail)
    ((let* ((el1 (car pat1))
            (el2 (car pat2))
            (new-bindings (if (var? el1)
                              (unify-var el1 el2 bindings)
                              (unify-const el1 el2 bindings))))
       (if (eq new-bindings :fail)
           :fail
           (unify (cdr pat1) (cdr pat2) new-bindings))))))

(defun unify-var (v el bindings)
  (let ((val (var-value v bindings)))
    (if (eq val :unbound)
        (cons (cons v el) bindings)
        (unify-const val el bindings))))

(defun unify-const (const el bindings)
  (if (var? el)
      (unify-var el const bindings)
      (if (eql const el) bindings :fail)))

(defun find-possible-unifiers (pattern)
  ;; Returns a list of rules and facts that might unify with pattern
  (if (var? (car pattern))
      *db*
      (append  (get '* 'database)
               (get (car pattern) 'database))))

(defun add-to-database (rule)
  ;; Returns NIL iff not added (because already present)
  (let ((index (index-of rule)))
    (cond
      ((member rule (get index 'database) :test #'equalp)
       nil)
      (t
       (push rule (get index 'database))
       (push rule *db*)
       t))))

(defun remove-from-database (rule)
  ;; Returns NIL iff not removed (because not present)
  (let* ((index (index-of rule))
         (the-rule (car (member rule (get index 'database) :test #'equalp))))
    (cond
      (the-rule
       (setf (get index 'database) (delete the-rule (get index 'database)
                                           :test #'eq))
       (setq *db* (delete the-rule *db* :test #'eq))
       t)
      (t nil))))

(defun index-of (rule)
  (caar (rule-consequents rule)))

(defun var? (thing)
  (and (symbolp thing) (char= (char (symbol-name thing) 0) #\*)))

(defun var-value (var bindings)
  ;; Follows the chain of variables bindings to find the ultimate value of var.
  (let ((val-pair (assoc var bindings)))
    (if (not val-pair)
        :unbound
        (let ((val (cdr val-pair)))
          (if (var? val)
              (var-value val bindings)
              val))))))

(defun rename-rule (rule)
  ;; Returns a list (c . a) of consequent and antecedents, with all
  ;; variables renamed.
  (multiple-value-bind (ants bindings)
      (copy-pattern-list (rule-antecedents rule) nil)
    (multiple-value-bind (conseqs ignorable-bindings)
        (copy-pattern-list (rule-consequents rule) bindings)
      (cons (car conseqs) ants))))

(defun copy-pattern-list (pat-list bindings)
  (if (null pat-list)
      (values nil bindings)
      (multiple-value-bind (new-pat binds)
          (copy-pattern (car pat-list) bindings)
        (multiple-value-bind (rest-pat final-binds)
            (copy-pattern-list (cdr pat-list) binds)
          (values (cons new-pat rest-pat) final-binds)))))

(defun copy-pattern (pat bindings)
  (do* ((els pat (cdr els))
```

(continued on page 62)

NEW! TOOLKITS FOR TURBO C & QUICK C from ZORTECH INC.



HOTKEY

A complete set of Terminate Stay Resident (TSR) functions that help you to write reliable 'pop-up' programs.

Now you can make your programs 'Sidekickable'. Two example programs are included, a 'pop-up Calculator' and a pop-up 'Critical Error Handler'.

The Hotkey toolkit handles all floating point functions in resident mode.

The 32 page manual includes an interesting discussion of the origin and history of undocumented MS-DOS function calls, together with a full explanation of the theory and practical use of TSR's.

Only $49.95! State Turbo C or Quick C version.)

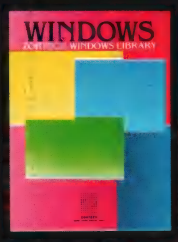
COMMS

Do you need to incorporate serial communications into your applications? Yes! Then get this inexpensive but highly professional COMMS toolkit from Zortech Inc.

Look at the list of features: Xmodem, Kermit and ASCII file transfer, Hayes modem control, VT52, VT100 and ANSI terminal emulation, supports up to 8 serial ports, speeds up to 19.2k baud rate and higher.

Two demonstration programs are included, MINICOM and MAXICOM (like Procomm) together with the 120 page manual and full source code FREE!

Only $49.95! (State Turbo C or Quick C version.)



GAMES

Have you ever wondered how to write a chess program? Now we reveal the secret algorithms and techniques of the masters with this dynamic Games toolkit.

The package comes complete with the full source code to three ready to play games of strategy – Chess, Backgammon and Wari (an ancient African game).

A comprehensive 150 page manual is provided giving an in depth look at the history, structure and program design of such 'Strategy Games'.

Only $49.95!

(State Turbo C or Quick C version.)

SUPERTEXT

This is not simply an 'Editor' toolkit, but a full-blown, 'WordStar' compatible wordprocessor with the full source code.

As well as all the normal editing functions, you will also find 'dot' commands and full printer control. The SuperText toolkit handles files of any size and allows full on-screen configuration.

Do you need to incorporate a wordprocessor into your application? Yes! Then get the SuperText toolkit complete with full source code and 150 page manual now!

Only $49.95! (State Turbo C or Quick C version.)

PROSCREEN

Generate high quality data entry screens with the Pro-Screen – Screen Designer and Code Generator.

You can draw the data entry screen, define the input fields, define the input criteria, set screen colors and attributes, draw single or double lines, make boxes – press a few buttons and 'hey presto' Pro-Screen generates the C source code for your application!

Professional applications programmers will find this versatile utility and it's associated functions invaluable.

Comes complete with a substantial 78 page manual and demo programs.

Only $49.95! (State Turbo C or Quick C version.)

ONLY $49.95 EACH

WINDOWS

Add super-fast text screen handling to your applications with the WINDOWS library from Zortech Inc.

Give your applications the professional look – with instant zooming and exploding windows. Incorporate drop-down menus and Lotus style menus with our easy to use functions.

Automatically handles memory saving and buffering of window text. Use any number of overlapping windows in your applications. Write to any window, read from any window, close any window, pull any window to the top.

Over 55 functions together with a big 85 page manual and remember, you get the full source code.

Only $49.95! (State Turbo C or Quick C version.)

NEW! C VIDEO

- *Now learn C the easy way!*
- *Get the 'Complete C Video Course' from Zortech Inc. together with our big 365 page workbook.*
- *Ten 1 hour tapes – 36 lessons!*
- *Easy to follow course, you get an excellent introduction to the C language.*
- *Takes you step-by-step up to the intermediate and advanced levels.*
- *Teach yourself at home or the office – at your own speed.*

~~only $295.00!~~ **$199.95**

Yes! Rush me these items!

☐ HOTKEY
☐ COMMS
☐ PRO-SCREEN
☐ WINDOWS ☐ GAMES
☐ SUPERTEXT ☐ C VIDEO

FREE SHIPPING – VISA/MC/COD/CHECK

Name .....

Address .....

..... Phone .....

VISA or MC# ..... Exp. Date .....

ZORTECH Inc. 361 Massachusetts Ave, Arlington, MA 02174

Support & Enquiries Tel: (617) 646-6703

ORDER HOTLINE (800) 848-8408

ZORTECH
BOSTON LONDON FRANKFURT GENEVA

Listing Two *(Listing continued, text begins on page 18.)*

```
        (el (car els) (car els))
        (new-pat nil))
       ((null els) (values (nreverse new-pat) bindings))
    (if (not (var? el))
        (push el new-pat)
        (let ((existing-var-pair (assoc el bindings)))
          (if existing-var-pair
              (push (cdr existing-var-pair) new-pat)
              (let ((new-var (rename-var el)))
                (push new-var new-pat)
                (push (cons el new-var) bindings)))))))

(defun rename-var (var)
  ;; Generate a new symbol.
  (gentemp (symbol-name var)))
```

End Listing Two

Listing Three

```
Listing Three. Code for streams

(defvar *empty-stream* nil)

(defmacro delay (thing)
  `#'(lambda () ,thing))

(defun force (thing)
  (funcall thing))

(defmacro stream-cons (thing stream)
  `(cons ,thing (delay ,stream)))

(defun stream-empty? (stream)
  (eq stream *empty-stream*))

(defun stream-car (stream)
  (car stream))

(defun stream-cdr (stream)
  (force (cdr stream)))

(defmacro dostream ((var stream) &body body)
  (let ((tempvar (gensym)))
    `(do* ((,tempvar ,stream (stream-cdr ,tempvar))
           (,var (stream-car ,tempvar) (stream-car ,tempvar)))
          ((stream-empty? ,tempvar) *empty-stream*)
       ,@body)))

(defmacro stream-append (stream1 stream2)
  `(stream-append-func ,stream1 (delay ,stream2)))


(defun stream-append-func (stream delayed-stream)
  (if (stream-empty? stream)
      (force delayed-stream)
      (stream-cons (stream-car stream)
                   (stream-append-func (stream-cdr stream) delayed-stream))))

(defun stream-mapcar (function stream)
  (if (stream-empty? stream)
      *empty-stream*
      (stream-cons (funcall function (stream-car stream))
                   (stream-mapcar function (stream-cdr stream)))))

(defun stream-mapcan (function stream)
  (if (stream-empty? stream)
      *empty-stream*
      (stream-append (funcall function (stream-car stream))
                     (stream-mapcan function (stream-cdr stream)))))

(defun stream->list (stream)
  (if (stream-empty? stream)
      nil
      (cons (stream-car stream)
            (stream->list (stream-cdr stream)))))

(defun list->stream (list)
  (if (null list)
      *empty-stream*
      (stream-cons (car list)
                   (list->stream (cdr list)))))
```

End Listing Three

Listing Four

```
Listing Four.  Code for the every action

(defun do-every-action (rule bindings)
  ;; Get a list of bindings for the single quantified variable, using the
  ;; antecedents; then execute the consequents for each binding.
  (let* ((quant-vars (rule-quant-vars rule)))
    (if (not (= (length quant-vars) 1))
        (error "Only one quantified variable allowed in rule a" rule)
```

```
(let* ((bindings-stream (deduce (rule-antecedents rule) bindings))
       (bindings-list (stream->list bindings-stream))
       (filtered-list (mapcar #'(lambda (b) (extract-bindings b
                                                quant-vars))
                              bindings-list))
       (undup-list (delete-duplicate-bindings filtered-list))
       (new-bindings-list (mapcar #'(lambda (b) (append b bindings))
                                  undup-list)))
  (dolist (new-bindings new-bindings-list)
    (do-rule-actions (rule-consequents rule) new-bindings))))))
```

End Listing Four

Listing Five

```
Listing Five.  The check-reqs function

(defun check-reqs (reqs bindings)
  (if (null reqs)
      t
      (let* ((req (car reqs))
             (binding-stream (deduce-pattern (requirement-pattern req)
                                             bindings))
             (fstring nil))
        (cond
          ((stream-empty? binding-stream)
           (return-from check-reqs (if (requirement-succeeded? req)
                                       nil
                                       (requirement-failure-string req))))
          (t
           (setf (requirement-succeeded? req) t)
           (dostream (binds binding-stream)
             (let ((result (check-reqs (cdr reqs) binds)))
               (if (eq result t)
                   (return-from check-reqs t)
                   (if result
                       (setq fstring result)))))
           fstring)))))
```

End Listings

SEMANTIC RESOLUTIONS

Listing One *(Text begins on page 50.)*

```
/*      A Theorem Prover for Propositional Logic
 *
 *      This program uses the resolution principle restricted to
 *  semantic clashes to prove theorems in propositional logic.
 *  The premises and the negation of the conclusion must be entered
 *  using 1, 0, and -1 to represent, respectively, the presence of a
 *  variable in true form, the absence of a variable, and the
 *  presence of a variable in negated form.  For example, a run
 *  in which the following four clauses are used
 *
 *       p v *q v  r
 *            q
 *      *p v *q
 *                *r
 *
 *  would look as follows:
 *
 *  enter number of clauses and variables
 *  4 3
 *  enter clauses
 *  1  -1  1
 *  0   1  0
 * -1  -1  0
 *  0   0 -1
 *
 *  The program would then respond with
 *
 *  initial clauses
 *    1 -1  1       (1)
 *    0  1  0       (2)
 *   -1 -1  0       (3)
 *    0  0 -1       (4)
 *  resolvents
 *    0 -1  0       (5 from 1, 3, 4)
 *    0  0  0       (6 from 2, 5)
 *  proof complete--empty clause generated
 *
 *      For each nucleus, the program searches for a set of electrons
 *  that make up a semantic clash.  When it finds a semantic clash, it
 *  generates the resolvent and then uses a recursive technique called
 *  backtracking to find other sets of electrons that also make up
 *  a semantic clash.  This implementation is compact but quite
 *  slow.
 *
 *  Author:  A. J. Dos Reis
 *           Dept. of Mathematics and Computer Science
 *           College at New Paltz
 *           New Paltz, N.Y. 12561
 *-------------------------------------------------------------------------
 *      CONSTANTS
 */
#define TRUE 1
#define FALSE 0
#define MAXCLAUSES 100
#define MAXVARS 10
#define ELECTRON -1
#define NUCLEUS 1
/*-------------------------------------------------------------------------
 *      GLOBAL VARIABLES
 */
int clause[MAXCLAUSES][MAXVARS];      /* Holds clauses                */
int clausetype[MAXCLAUSES];           /* Holds type:  nuc or elec     */
int clashelec[MAXVARS];               /* Holds row numbers of electrons
                                         that form a semantic clash   */
int nvars;                            /* Number of variables          */
int avail;                            /* Index of next avail row      */
int newclauses;                       /* TRUE if new resolvents
                                         generated on last pass       */
int noemptyclause;                    /* TRUE if empty clause not
                                         generated                    */
/*-------------------------------------------------------------------------
 *      main prompts for and inputs the clauses.  It then calls
 *      findsemclash for each nucleus.  It continues until the
 *      empty clause is generated or until no new resolvents are
 *      generated.
 */
main()
{
   int nclauses;                      /* Number of clauses            */
   int nuc;                           /* Index of nucleus             */
   int i,j;                           /* Utility indices              */
   printf("enter number of clauses and variables\n");
   scanf("%d%d",&nclauses,&nvars);
   printf("enter clauses\n");
   for (i=0; i<nclauses; i++)         /* Read in clauses              */
```

Listing One *(Listing continued, text begins on page 50.)*

```
      {clausetype[i]=ELECTRON;
       for (j=0; j<nvars; j++)
          {scanf("%d",&clause[i][j]);
           if (clause[i][j]==1)
              clausetype[i]=NUCLEUS; /* Set clausetype accordingly  */
          }
       }
   avail=nclauses;                   /* Keep track of next avail row*/
   printf("initial clauses\n");      /* Print out initial clauses   */
   for (i=0; i<nclauses; i++)
      {for (j=0; j<nvars; j++)
         printf("%3d",clause[i][j]);
       printf("     (%1d)\n",i+1);
      }
   printf("resolvents\n");
   noemptyclause=TRUE;               /* Initialize flag             */
   do
      {newclauses=FALSE;             /* Initialize flag             */
       nuc=0;                        /* Start search from row 0     */
       do
          {if (clausetype[nuc]==NUCLEUS) /* Search for nucleus       */
             {for (i=0; i<nvars; i++) /* Initialize clashelec        */
                 clashelec[i]=-99;
              findsemclash(0,nuc);   /* Look for all sem clashes    */
             }
           nuc++;                    /* Repeat for next clause      */
          } while ((nuc<nclauses) && noemptyclause);
      } while (newclauses && noemptyclause);
   if (noemptyclause)                /* Failed to gen empty clause? */
      printf("argument not valid\n");
   else
      printf("proof complete--empty clause generated\n");
}
/*---------------------------------------------------------------------
 *     findsemclash searches for an electron which under semantic
 *     resolution will eliminate the true literal in column truecol
 *     of the nucleus in row nuc.  When it finds a satisfactory
 *     electron, it recursively calls itself to find a electron that
 *     eliminates the next true literal.  When a set of electrons
 *     that forms a semantic clash is found, findsemclash generates
 *     the resolvent instead of making a recursive call.  Then
 *     it backtracks to find other sets that also form semantic clashes.
 */
findsemclash(truecol,nuc)
{int col;                            /* Column index                */
 int elec;                           /* Row number of electron      */
 int goodelec;                       /* True if elec ok under
                                        semantic resolution         */
 int i,j;                            /* Utility indices             */
 while (truecol<nvars)               /* Checked all columns of nuc? */
   if (clause[nuc][truecol]==1)      /* Is this a true literal      */
      {elec=0;                       /* Search for appropriate elec */
       do
          {if (clausetype[elec]==ELECTRON)
              {col=0;                /* check col-by-col if elec ok */
               goodelec=TRUE;
               while ((col<nvars) && goodelec)
                  {if (               /* Satisfies sem resolution?  */
                        ( (col<truecol) && (clause[elec][col]!=0))
                                      ||
                        ( (clause[nuc][col] * clause[elec][col]==-1)
                                      !=
                          (truecol==col))
                      )
                     goodelec=FALSE; /* Electron fails test         */
                   col++;             /* Test next column           */
                  }
               if (goodelec)         /* Electron passed test?       */
                  {clashelec[truecol]=elec; /* Remember row number  */
                   findsemclash(truecol+1,nuc); /* Recursive call   */
                  }
              }
            elec++;                  /* Try next electron           */
          } while ((elec<avail) && noemptyclause);
       return;                       /* Backtrack                   */
      }
   else
      truecol++;                     /* Continue search for true lit
                                        in the nucleus              */
 newclauses=TRUE;                    /* Get here only when a sem
                                        clash is found              */
 for (i=0; i<nvars; i++)             /* Copy nuc to new avail row   */
   clause[avail][i]=clause[nuc][i];
 for (i=0; i<nvars; i++)
   {elec=clashelec[i];               /* Get row number of electron  */
```

(continued on next page)

Listing One *(Listing continued, text begins on page 50.)*

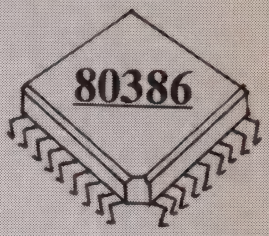
```
     if (elec!=-99)                    /* -99 means electron not needed
                                          for the ith col            */
        for (j=0; j<nvars; j++)        /* Generate the resolvent      */
           {clause[avail][j]+=clause[elec][j];
            if (clause[avail][j]!=0)
               noemptyclause=TRUE;
            if (clause[avail][j]==-2)
                clause[avail][j]=-1;
           }
    }
 noemptyclause=FALSE;                  /* Initialize flag             */
 for (j=0; j<nvars; j++)               /* Check for empty clause      */

    {printf("%3d",clause[avail][j]);
     if (clause[avail][j])
        noemptyclause=TRUE;            /* Reset if no empty clause    */
    }
 printf("     (%1d from %1d",avail+1,nuc+1); /* Print resolvent       */
 for (i=0;i<nvars; i++)
    if (clashelec[i]!=-99)             /* Print electron row numbers */
          printf(", %1d",clashelec[i]+1);
 printf(")\n");
 clausetype[avail]=ELECTRON;           /* Set type of resolvent       */
 avail++;                              /* Increment avail row index   */
 }
```

End Listing

SERIOUS DEBUGGING AT A REASONABLE PRICE

NEW

Soft-ICE™



All the speed and power of a hardware-assisted debugger at a software price - $386

Features

Real-time break points on:
- Memory locations
- Memory ranges
- Execution
- I/O ports
- Interrupts (hardware and software)

Dual monitor support

System memory map

Regain control with a keystroke

Even with the following code:

```
CLI
MOV   AL,0FFH
OUT   21H,AL
JMP   $
```

and much, much more

How Soft-ICE works

Soft-ICE unleashes the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment. Soft-ICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to add real-time hardware-level breakpoints to your existing DOS debugger. To use Soft-ICE you simply pop the Soft-ICE window up through a key sequence, set your hard break points, then return to your soft debugger. As the target program is executing, Soft-ICE recognizes when the breakpoint conditions have been reached and gives control back to your soft debugger. And this is all done at full 80386 speed! Soft-ICE can also be used in stand-alone mode. This comes in handy if you are debugging loadable device drivers, interrupt handlers, or terminate and stay resident programs. All of the standard debugging features are available to help you find the most difficult systems problems.

Benefits

- **Works with CodeView** --To get you up and going as fast as possible, Soft-ICE is designed to work with your existing software debuggers such as CodeView and Periscope I & II.
- **Breaks the 640K barrier** -- If you have extended memory, Soft-ICE takes up **ZERO** bytes of memory in the first 1MB of address space. This means you can load and debug your largest programs.
- **Power of an in-circuit emulator** -- At only $386, you can give every member of your software development team the power of an in-circuit emulator.
- **AT compatible 80386 PC's** -- Soft-ICE works with all AT compatible 80386 PC's, such as COMPAQ's Deskpro 386 and the IBM Model 80.
- **Easy to learn** -- Soft-ICE is so easy to learn that you can be finding bugs with Soft-ICE by the time you could install a hardware-assisted debugger.

"Since Soft-ICE doesn't take up any of my memory I have it in my AUTOEXEC.BAT to load every day... It has saved me at least one month's time on my latest device driver program." Peter Ricker, President of Maverick Software

30 day money-back satisfaction guarantee. Visa and Master Card accepted. Ask about our coupon program.

To order or to get more information, call (603) 888-2386

NU-MEGA TECHNOLOGIES

P.O. BOX 7607
NASHUA, NH 03060-7607

CIRCLE NO. 149 ON READER SERVICE CARD

Listing One *(Text begins on page 98.)*

```
#include <stdarg.h> /* ANSI variable-argument defines          */
#include <dos.h>    /* uSoft: defines for FP_OFF and FP_SEG    */

/* IDOPRNT - integer-only printf/fprintf/sprintf workhorse
 *           function.
 *
 *           (C) 1988 Allen I. Holub. All rights reserved.
 *
 * The following conversions are supported:
 *
 *      %d %ld  decimal, long decimal
 *      %u      unsigned (int only, no longs)
 *      %s      string
 *      %x %lx  hex, long hex
 *      %o %lo  octal, long octal
 *      %b %lb  binary, long binary                    (nonstandard)
 *      %p      far pointer (in hex XXXX:XXXX)
 *
 *      Note that it's impossible to get zero fill in the offset
 *      part of a %p conversion. That is 1234:0001 is always
 *      printed as 1234:1. Sorry. The following modifiers are
 *      supported for all conversions:
 *
 *      %0x     zero left fill
 *      %-x     left justification in field
 *      %|x     centered in field               (nonstandard)
 *      %*x     field width from first argument
 *
 *      Precision is supported in strings:
 *
 *      %X.Ys   X-character wide field, print at most Y characters
 *              (even if the string is longer). If X or Y is a *,
 *              the width is taken from the next argument.
 *
 *      Potential portability problems:
 *
 *      %p is a FAR pointer. I've used the "far" keyword to
 *      declare it as such and I've used the FP_SEG and FP_OFF
 *      to extract the segment and offset parts of the pointer.
 *      The offset part of a near pointes can be printed by
 *      casting it to an int and using %x.
 */

extern char     *ltos(long, char*, int);

/*------------------------------------------------------------
 * Macros to save some code space below. If your compiler
 * doesn't accept multi-line macros, remove the backslashes
 * and merge the entire definition onto one line.
 * These both have horrible side effects. Be careful.
 *
 * PAD(fw,filchar,out,op)   outputs filchar, fw times. fw = 0.
 * TOINT(p,x)               works like "x = atoi(p);" except p
 *                              is advanced past the number.
 *
 */

#define PAD(fw,fc,out,op)     while( --(fw) >= 0 )        \
                                  (*out)( fc, op )

#define TOINT(p,x)  while( '0' <= *p  &&  *p <= '9' )   \
                        x = (x * 10) + (*p++ - '0')

/*------------------------------------------------------------
 * INTMASK is a portable way to mask off the bottom N bits
 * of a long, were N is the width of an int.
 */

#define INTMASK         (long)( (unsigned)(0) )

/*------------------------------------------------------------*/

void    idoprnt( out, o_param, format, args )

int     (*out)();       /* output subroutine                */
void    *o_param;       /* 2nd argument to pass to out()    */
char    *format;        /* pointer to format string         */
va_list args;           /* pointer to arguments             */
{
    char filchar ;      /* Fill character used to pad fields */
    char nbuf[34];      /* Buffer used to hold converted #s  */
    char *bp     ;      /* Pointer to current output buffer  */
    int slen     ;      /* length of string pointe to by bp  */
    int base     ;      /* Current base (%x=16, %d=10, etc.) */
    int fldwth   ;      /* Field width as in %10x            */
    int precision;      /* Precision as in %10.10s or %10.3f */
    int lftjust  ;      /* 1 = left justifying (ie. %-10d)   */
    int center   ;      /* 1 = centered (ie. %|10d)          */
    int longf    ;      /* doing long int (ie. %lx or %X)    */
    long lnum    ;      /* used to hold numeric arguments    */
    void far *pnum;     /* Pointer-sized number              */

```

```
 92|     for(; *format ; format++ )
 93|     {
 94|         if( *format != '%')             /* No conversion, just  */
 95|         {                               /* print the next char. */
 96|             (*out)(*format, o_param);
 97|         }
 98|         else                            /* Process a % conversion */
 99|         {
100|             bp          = nbuf  ;
101|             filchar      = ' '   ;
102|             fldwth       = 0     ;
103|             lftjust      = 0     ;
104|             center       = 0     ;
105|             longf        = 0     ;
106|             precision    = 0     ;
107|             slen         = 0     ;
108|
109|             /*  Interpret any modifiers that can precede a
110|              *  conversion character. (ie %04x , %-10.6s... etc).
111|              *  if a * is present instead of a field width then
112|              *  the width is taken from the argument list.
113|              */
114|
115|             if( *++format == '-') { ++format ;  ++lftjust;      }
116|             if( *format   == '|') { ++format ;  ++center;       }
117|             if( *format   == '0') { ++format ;  filchar = '0'; }
118|
119|             if( *format != '*' )
120|                 TOINT( format, fldwth );
121|             else
122|             {
123|                 ++format;
124|                 fldwth = va_arg(args, int);
125|             }
126|
127|             if( *format == '.' )
128|             {
129|                 if( *++format != '*' )
130|                     TOINT( format, precision );
131|                 else
132|                 {
133|                     ++format;
134|                     precision = va_arg(args, int);
135|                 }
136|             }
137|
138|             if( *format == 'l' || *format == 'L' )
139|             {
140|                 ++format;
141|                 ++longf ;
142|             }
143|
144|
145|             /* By now we've picked off all the modifiers and
146|              * *format is looking at the actual conversion
147|              * character. Pick the appropriatly sized argument
148|              * off the stack and advanced the pointer (ap) to
149|              * point at the next argument.
150|              */
151|
152|             switch( *format )
153|             {
154|             default:    *bp++ = *format ;               break;
155|             case 'c':   *bp++ = va_arg(args, int   );   break;
156|             case 's':   bp    = va_arg(args, char *);   break;
157|
158|             case 'p':
159|                 pnum  = va_arg(args, void far *);
160|                 bp    = ltos( (unsigned long)FP_SEG(pnum), bp, 16);
161|                 *bp++ = ':';
162|                 bp    = ltos( (unsigned long)FP_OFF(pnum), bp, 16 );
163|                 break;
164|
165|             case 'u':   base = -10 ; goto pnum ;
166|             case 'd':   base = 10  ; goto pnum ;
167|             case 'x':   base = 16  ; goto pnum ;
168|             case 'b':   base =  2  ; goto pnum ;
169|             case 'o':   base =  8  ;
170| pnum:
171|                 /* Fetch a long or int sized argument off the
172|                  * stack as appropriate. If the fetched number
173|                  * is a base 10 int then mask off the top
174|                  * bits to prevent sign extension.
175|                  */
176|
177|                 if( longf )
178|                     lnum = va_arg(args, long);
179|                 else
180|                 {
181|                     lnum = (long) va_arg(args, int);
182|
183|                     if( base == -10 )       /* Unsigned int */
184|                     {
185|                         base  = 10;
186|                         lnum &= INTMASK;
187|                     }
```

(continued on next page)

Listing One *(Listing continued, text begins on page 98.)*

```
188|                    else if( base != 10 )   /* Nondecimal int */
189|                    {
190|                        lnum &= INTMASK;
191|                    }
192|                }
193|
194|                if( lnum < 0L  &&  base == 10  && filchar == '0' )
195|                {
196|                        /* Again, print a - to avoid "000-123."
197|                         * Only decimal numbers get - signs.
198|                         */
199|
200|                        (*out)( '-' ,o_param) ;
201|                        --fldwth ;
202|                        lnum = -lnum ;
203|                }
204|
205|                bp   = ltos( lnum, bp, base );
206|                break;
207|            }
208|
209|            /*  Terminate the string if necessary and compute
210|             *  the string length (slen). Bp will point at the
211|             *  beginning of the output string.
212|             */
213|
214|            if (*format != 's')
215|            {
216|                *bp  = '\0';
217|                slen = bp - nbuf;
218|                bp   = nbuf;
219|            }
220|            else
221|            {
222|                slen = strlen(bp);
223|                if( precision  &&  slen > precision )
224|                        slen = precision;
225|            }
226|
227|
228|            /*  Adjust fldwth to be the amount of padding we need
229|             *  to fill the buffer out to the specified field
230|             *  width. Then print leading padding (if we aren't
231|             *  left justifying), the buffer itself, and any
232|             *  required trailing padding (if we are left
233|             *  justifying.
234|             */
235|
236|            if( (fldwth -= slen) < 0 )
237|                fldwth = 0;
238|
239|            if( center )
240|            {
241|                                /* Use longf as counter */
242|                longf = fldwth/2;
243|                PAD( longf, filchar, out, o_param );
244|            }
245|            else if( !lftjust )
246|                PAD( fldwth,   filchar, out, o_param );
247|
248|            while( --slen >= 0 )
249|                (*out)(*bp++,o_param);
250|
251|            if( center )
252|            {
253|                longf = fldwth - fldwth/2;
254|                PAD( longf, filchar, out, o_param );
255|            }
256|            else if( lftjust)
257|                PAD( fldwth, filchar, out, o_param );
258|
259|        }
260|    }
261| }
262|
263| /*------------------------------------------------------------*/
264|
265| #ifdef DEBUG
266|
267| #include <stdio.h>
268|
269| printm( fmt, ... )
270| char    *fmt;
271| {
272|         extern int  fputc();
273|         va_list args;
274|         va_start(args, fmt);
275|         idoprnt( fputc, stdout, fmt, args );
276| }
277|
278| /*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
279|
280| main()
281| {
```

```
282|     printm("should see <0 1 2 3 4 5 6 7>: ");
283|     printm("          %d %x %o %ld %lx %lo %c %s\n",
284|                           0, 1, 2, 3L, 4L, 5L, '6', "7" );
285|
286|     printf("should see: hello world: "          );
287|     printm("%s %s %c", "hello", "world", '\n'   );
288|
289|     printm("should see <string>  : <%6.6s>\n",  "string NO NO" );
290|     printm("should see <   str>  : <%*.*s>\n",  6, 3, "string" );
291|     printm("should see <70000>   : <%ld>\n", 70000L     );
292|     printm("should see <fffff>   : <%lx>\n",0xfffffL);
293|     printm("should see <ffff>    : <%x>\n",      -1      );
294|     printm("should see <-1>      : <%ld>\n",     -1L     );
295|     printm("should see <x>       : <%c>\n",      'x'     );
296|     printm("should see <  x  >   : <%|5c>\n",    'x'     );
297|     printm("should see <a5>      : <%x>\n",      0xa5    );
298|     printm("should see <765>     : <%o>\n",      0765    );
299|     printm("should see <1010>    : <%b>\n",      0xa     );
300|     printm("should see <   123>  : <%6d>\n",     123     );
301|     printm("should see <   456>  : <%*d>\n",   6,456     );
302|     printm("should see <  -123>  : <%6d>\n",    -123     );
303|     printm("should see <123   >  : <%-6d>\n",    123     );
304|     printm("should see <-123  >  : <%-6d>\n",   -123     );
305|     printm("should see <-00123>  : <%06d>\n",   -123     );
306|     printm("should see <abcd:123>: <%p>\n",
307|                                                 (char far *)( 0xabcd0123L) );
308| }
309| #endif
```

End Listing One

Listing Two

```
 1| /* LTOS.C        Convert long to string in indicated base.
 2|  *               (C) 1988 Allen I. Holub. All rights reserved.
 3|  */
 4|
 5| char    *ltos( n, buf, base )
 6| unsigned long   n    ;
 7| char            *buf ;
 8| int             base ;
 9| {
10|     /*  Convert long to string. Prints in hex, decimal, octal,
11|      *                                          or binary.
12|      *  "n"    is the number to be converted
13|      *  "buf"  is the output buffer.
14|      *  "base" is the base (16, 10, 8 or 2).
15|      *
16|      *  The output string is null terminated and a pointer to the
17|      *  null terminator is returned.
18|      *
19|      *  The number is put into an array one digit at a time as
20|      *  it's translated. The array is filled with the digits
21|      *  reversed (ie. the \0 goes in first, then the rightmost
22|      *  digit, etc.) and then is reversed in place before
23|      *  returning.
24|      *
25|      *  This routine is much like the unix() ltoa except that
26|      *  it returns a pointer to the end of the string.
27|      */
28|
29|     register char        *bp   = buf;
30|     register int         minus = 0;
31|     char                 *endp;
32|
33|     if( base < 2 || base > 16 )
34|             return 0;
35|
36|     if( base == 10 && (long)n < 0 ) /* If the number is negative */
37|     {                               /* and we're in base 10, set */
38|         minus++ ;                   /* minus to true and make it */
39|         n = -( (long)n );           /* positive.                 */
40|     }
41|
42|     *bp = '\0' ;              /* Have to put the null in now */
43|                               /* because the array is being  */
44|                               /* filled in reverse order.    */
45|     do {
46|             *++bp = "0123456789abcdef" [ n % base ];
47|             n /= base;
48|
49|     } while( n );
50|
51|     if( minus )
52|             *++bp = '-';
53|
54|     for( endp = bp; bp > buf ;)       /* Reverse string in place */
55|     {
56|             minus  = *bp;            /* Use minus for temporary */
57|             *bp -- = *buf;           /* storage.                */
58|             *buf++ = minus;
59|     }
60|
61|     return endp;        /* Return pointer to terminating null   */
62| }
```

End Listings

TO THE MACS

Listing One *(Text begins on page 106.)*

```
-------------------------------------------------------

Scouting Toolkit™
A HyperCard Project

Programmed and ©1988 by Stan Krute
All rights reserved

-------------------------------------------------------

Description Of Objects

-------------------------------------------------------

The Stack
   Description
      The stack's name is "Scouting Toolkit™"
      It contains 1 background
      It contains 1 card
      The size of the stack is 20K
   Script
      The stack has no script

-------------------------------------------------------

Background 1
   Description
      The background has no name
      It's used by 1 card
   Script
      The background has no script

-------------------------------------------------------

Card 1
   Description
      The card's name is "Scouting Toolkit"
      It contains 17 buttons
      It contains 2 fields
   Script
      ---------------------------- openCard -------------------------
      on openCard
        -- hide the menubar, and initialize a state variable for it
        global menubarState
        hide menubar
        put "hidden" into menubarState

        -- hide some windows
        hide message box
        hide tool window
        hide pattern window
      end openCard

      ---------------------------- mouseUp --------------------------
      on mouseUp
        -- redraw the toolkit buttons, in case some are hidden
        repeat with buttonNumber = 2 to 13
          show button buttonNumber
        end repeat
      end mouseUp
-------------------------------------------------------

Button 1
   Description
      The button's name is "Delete Scouting Toolkit"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30010
      Its rectangle is 42 units wide, 42 unitshigh
      The button's located at card location (36,292)
```

```
   Its text properties are: System Font 12, Height 16, Align center
Script
   ------------------------------ mouseUp ------------------------------
   on mouseUp
     -- make sure we don't mess up the supply card
     if short name of this card is not "Scouting Toolkit" then

       -- save and set the user level
       get userLevel
       put it into entryUserLevel
       set userLevel to 5

       -- show the watch cursor
       set cursor to 4

       -- make sure the kit buttons are showing
       show button "Scouting Toolkit Icon"
       click at loc of button "Scouting Toolkit Icon"

       -- set the tool
       choose button tool

       -- get rid of our toolkit
       click at loc of button "Home"
       doMenu "Clear Button"

       click at loc of button "Back"
       doMenu "Clear Button"

       click at loc of button "Next"
       doMenu "Clear Button"

       click at loc of button "Previous"
       doMenu "Clear Button"

       click at loc of button "Toggle Pattern Window"
       doMenu "Clear Button"

       click at loc of button "Toggle Tool Window"
       doMenu "Clear Button"

       click at loc of button "Toggle Message Box"
       doMenu "Clear Button"

       click at loc of button "Toggle Menubar"
       doMenu "Clear Button"

       choose field tool
       show card field "Copyright"
       click at loc of card field "Copyright"
       doMenu "Clear Field"
       choose button tool

       click at loc of button "Scouting Toolkit About"
       doMenu "Clear Button"

       click at loc of button "Scouting Toolkit™"
       doMenu "Clear Button"

       click at loc of button "Scouting Toolkit Window"
       doMenu "Clear Button"

       show button "Scouting Toolkit Icon"
       click at loc of button "Scouting Toolkit Icon"
       doMenu "Clear Button"

       -- snake swallows tail
       click at loc of button "Delete Scouting Toolkit"
       doMenu "Clear Button"

       -- set the tool
       choose browse tool

       -- restore the cursor
       set cursor to 0
```

(continued on next page)

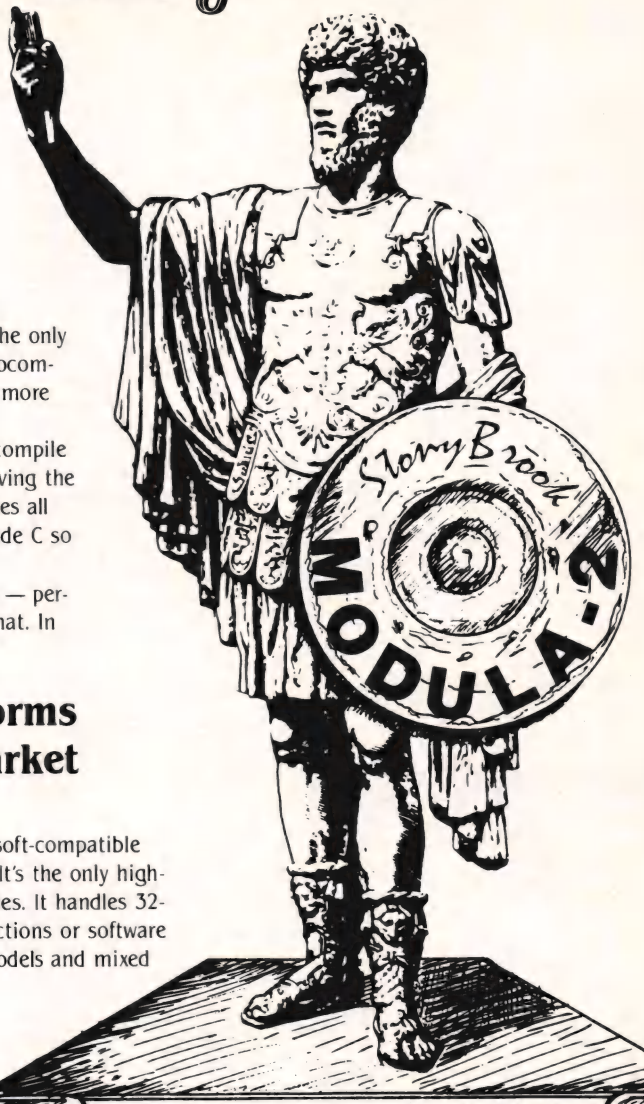
TO THE MACS

Listing One *(Listing continued, text begins on page 106.)*

```
        -- restore the user level
        set userLevel to entryUserLevel

      else
        -- we're on the supply card, so it would be bad form to delete

        -- send a signal
        show card field "No Delete Here"

        -- wait a moment
        wait 1 seconds

        -- hide the signal
        hide card field "No Delete Here"
      end if
    end mouseUp

-------------------------------------------------------------

Button 2
  Description
    The button's name is "Scouting Toolkit Window"
    Its Show name property is false
    Its Auto hilite property is true
    Its style is shadow
    It has no icon
    Its rectangle is 300 units wide, 182 units high
    The button's located at card location (197,144)
    Its text properties are: System Font 12, Height 16, Align center
  Script
    ---------------------------- mouseUp ----------------------------
    on mouseUp
      -- make a noise
      play "boing" "c"

      -- close the kit
      hideScoutingToolkit

      -- show the trigger
      show button "Scouting Toolkit Icon"
    end mouseUp


    ----------------------- hideScoutingToolkit -----------------------
    on hideScoutingToolkit
      -- hide the kit's buttons
      hide button "Home"
      hide button "Back"
      hide button "Next"
      hide button "Previous"
      hide button "Toggle Pattern Window"
      hide button "Toggle Tool Window"
      hide button "Toggle Message Box"
      hide button "Toggle Menubar"
      hide button "Scouting Toolkit About"
      hide button "Scouting Toolkit™"
      hide button "Scouting Toolkit Window"
    end hideScoutingToolkit

-------------------------------------------------------------

Button 3
  Description
    The button's name is "Scouting Toolkit™"
    Its Show name property is true
    Its Auto hilite property is true
    Its style is opaque
    It has no icon
    Its rectangle is 120 units wide, 20 units high
    The button's located at card location (296,160)
    Its text properties are: System Font 12, Height 16, Align center
  Script
    ---------------------------- mouseUp ----------------------------
    on mouseUp
      -- pass the message through
```

(continued on page 76)

I have come to bury C, sir, not to praise it.



C served us well in the days of kilobytes and kilohertz. It was the only language we could implement efficiently on our newborn microcomputers. But with today's mega-machines, shouldn't we demand more from our compilers?

Modula-2 increases productivity by catching your errors at compile time. You'll easily modularize and structure your programs, driving the hordes of barbaric bugs into the hinterlands. And Modula-2 does all this without taking away the low-level machine access that made C so popular.

Until now, you had to pay a price for the Modula-2 advantages — performance just didn't measure up to C. But we've changed all that. In a suite of benchmarks developed by PC Week:

Stony Brook Modula-2 outperforms the best C compilers on the market

(and no other Modula-2 compiler even comes close).

Stony Brook Modula-2, for 80x86 machines, produces Microsoft-compatible objects, and fully supports both Microsoft Windows and OS/2. It's the only high-level language compiler that lets you write dynamic link libraries. It handles 32- and 64-bit real numbers with in-line 80x87 coprocessor instructions or software emulation. And Stony Brook Modula-2 supports six memory models and mixed model programming.

You might want to bury your C compiler once you have used Stony Brook Modula-2, but you won't have to. We made it possible to directly call C and other languages from Modula-2, so you won't have to throw away your investment in C code.

So, friends, programmers, and C-users, lend us your ears. Call us or write for more information and to find out how you can get a demo compiler.

Benchmark	Stony Brook M-2 V1	Logitech M-2 V3	Turbo C V1	Microsoft C V4
Magnifier	0.05	0.06	0.05	0.05
For Loop	0.28	0.73	0.44	0.68
While Loop	0.35	0.75	0.49	0.66
Repeat Loop	0.33	0.68	0.38	0.66
Literal Assign	0.33	0.97	0.60	0.94
Memory Access	0.33	1.04	0.55	1.00
Real Arithmetic	4.68	13.40	3.73	2.96
Real Algebra	4.18	12.87	3.07	3.64
Vector	0.77	1.75	1.21	1.26
Equal If	0.66	1.47	0.93	1.32
Unequal If	0.66	1.43	0.99	1.32
No Parameters	0.33	0.60	0.34	1.04
Value	0.50	0.71	0.60	1.21
Reference	0.61	0.77	0.66	1.21
Real Maths	3.90	4.61	7.85	18.51

The compiler package includes DOS runtime library objects and full sources for our split-screen text editor for $195. The development package includes all of the above plus an automatic make utility, a symbolic debugger, and run-time library sources for $345. MasterCard and Visa accepted. Add $5 for shipping in North America, $25 for overseas shipping.

Stony Brook INC.
SOFTWARE

Forest Road, Wilton, New Hampshire 03086 ● (603)654-2525 Ask for Cleopatra.

CIRCLE NO. 156 ON READER SERVICE CARD

Listing One *(Listing continued, text begins on page 106.)*

```
        send mouseUp to button "Scouting Toolkit Window"
    end mouseUp

-------------------------------------------------------------

Button 4
  Description
    The button's name is "Scouting Toolkit About"
    Its Show name property is false
    Its Auto hilite property is true
    Its style is transparent
    It has an icon, ID# 30011
    Its rectangle is 24 units wide, 22 units high
    The button's located at card location (466,150)
    Its text properties are: System Font 12, Height 16, Align center
  Script
    ------------------------------ mouseUp -------------------------
    on mouseUp
      -- show the copyright notice
      show card field "Copyright"

      -- a little sound effect
      play "harpsichord" "c e g"

      -- wait for a mouse click
      wait until the mouseClick

      -- a little sound effect
      play "harpsichord" "g e c"

      -- hide the copyright notice
      hide card field "Copyright"
    end mouseUp

-------------------------------------------------------------

Button 5
  Description
    The button's name is "Toggle Menubar"
    Its Show name property is false
    Its Auto hilite property is true
    Its style is shadow
    It has an icon, ID# 30009
    Its rectangle is 42 units wide, 42 units high
    The button's located at card location (225,193)
    Its text properties are: System Font 12, Height 16, Align center
  Script
    --------------------------- mouseUp ---------------------------
    on mouseUp
      -- no built-in check for visibility, so…
      global menubarState

      -- toggle the menubar's visibility
      if menubarState is "hidden" then
        show menubar
        put "showing" into menubarState
      else
        hide menubar
        put "hidden" into menubarState
      end if
    end mouseUp

-------------------------------------------------------------

Button 6
  Description
    The button's name is "Toggle Message Box"
    Its Show name property is false
    Its Auto hilite property is true
    Its style is shadow
    It has an icon, ID# 30008
    Its rectangle is 42 units wide, 42 units high
    The button's located at card location (292,193)
    Its text properties are: System Font 12, Height 16, Align center
```

```
  Script
     ---------------------------- mouseUp ----------------------------
     on mouseUp
       -- toggle the message window's visibility
       if the visible of message box is true then
         hide the message box
       else
         show the message box
       end if
     end mouseUp
```

```
Button 7
   Description
      The button's name is "Toggle Tool Window"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30007
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (359,193)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- toggle the tool palette window's visibility
        if the visible of tool window is true then
          hide tool window
        else
          show tool window
        end if
      end mouseUp
```

```
Button 8
   Description
      The button's name is "Toggle Pattern Window"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30006
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (426,193)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- toggle the pattern palette's window's visibility
        if the visible of pattern window is true then
          hide pattern window
        else
          show pattern window
        end if
      end mouseUp
```

```
Button 9
   Description
      The button's name is "Previous"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30001
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (225,259)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- move to the previous card
```

(continued on next page)

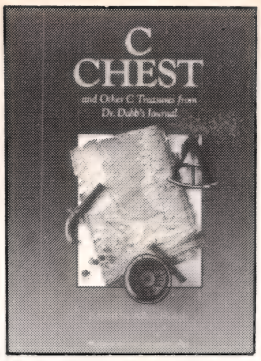
Listing One *(Listing continued, text begins on page 106.)*

```
      doMenu "Prev"
   end mouseUp
-------------------------------------------------------------

Button 10
   Description
      The button's name is "Next"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30002
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (292,259)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- move to the next card
        doMenu "Next"
      end mouseUp

-------------------------------------------------------------

Button 11
   Description
      The button's name is "Back"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30003
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (359,259)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- go to the last card visited
        doMenu "Back"
      end mouseUp

-------------------------------------------------------------

Button 12
   Description
      The button's name is "Home"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30005
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (426,259)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ---------------------------- mouseUp ----------------------------
      on mouseUp
        -- go home
        visual effect iris close
        doMenu "Home"
      end mouseUp

-------------------------------------------------------------

Button 13
   Description
      The button's name is "Scouting Toolkit Icon"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30004
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (178,83)
```

(continued on page 83)

The Best C Articles from *Dr. Dobb's Journal*

C Chest and Other C Treasures



by Allen Holub

This comprehensive anthology contains the popular "C Chest" columns from *Dr. Dobb's Journal of Software Tools,* along with the lively philosophical and practical discussions they inspired, plus other information-packed articles by C experts.

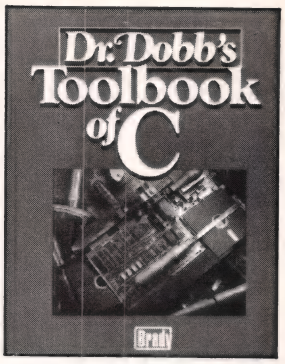
Topics covered include: pipes, wild-card expansion, and quoted arguments; sorting routines; command-line processing; queues and bit maps; utilities such as ls, make, and more; expression parsing; hyphenation; IBM cursor control and an Fget that edits; redirection; accessing IBM video display memory; trees; an AVL tree database package; directory traversal; sets; shrinking .EXE file images; hashing, expressions, and roman numerals; and statistical applications of digital low-pass filters.

Other treasures include: a variable metric minimizer; Fgrep; a peephole optimizer; and curve fitting with cubic splines.

All subroutines and programs are written in C, and are available on disk with full source code. MS-DOS format.

Book & Disk (MS-DOS)	**Item #49-6**	**$38.95**
Book	**Item #40-2**	**$23.95**

Dr. Dobb's Toolbook of C

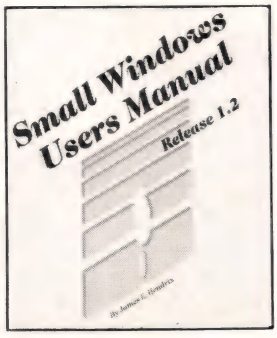


by the Editors of
Dr. Dobb's Journal
of Software Tools

This authoritative reference contains more than 700 pages of the best C articles and source code from Dr. Dobb's Journal of Software Tools, along with new material by C experts. The level is sophisticated and pragmatic: appropriate for professional C programmers. You'll find hundreds of pages of useful C source code, including a complete compiler, an assembler, and text- processing utilities. Highlights include:

- James E. Hendrix's famous Small-C Compiler and New Library for Small C
- Also, James E. Hendrix's Small Mac: An Assembler for Small C and Small Tools: Programs for Text Processing
- All of Anthony Skjellum's C Programmer's Notebook columns distilled into one thought-provoking chapter

Book	**Item #615-3**	**$29.95**



Small Windows: A Library of Windowing Functions for the C Language

by James E. Hendrix

Small-Windows is a complete windowing library for C (Microsoft 4.0/5.0, Turbo C, Lattice C, and Small-C). The package includes:

- 18 video functions written in assembly language
- 7 menu functions that support both static and pop-up menus
- 41 window functions to clean, frame, move, hide, show, scroll, push and pop windows

Complete Source Code Included — Microsoft 4.0/5.0, Turbo C, Lattice 3.0 and Small C are all supported

A file directory facility illustrates the use of window menu functions and provides file selection, renaming, and deletion capability. Two test programs are provided as examples to show you how to use the library and the window, menu, and directory functions.

The **Small-Windows** package is available for MS-DOS systems, and Microsoft C Version 4.0, Turbo C, Lattice C and Small-C compilers. Documentation and full C source code is included.

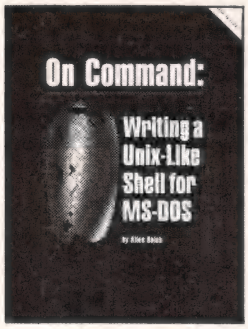
Manual & Disk (MS-DOS)
(Microsoft C, Small-C, Lattice C, or Turbo C Compiler)
Item #35-6 $29.95

TO ORDER: Return the postage paid card on the next page or CALL TOLL-FREE 800-533-4372 (In CA 800-356-2002)

**** Also available at your local bookstore***

Unix-Like Programming Tools for MS-DOS

On Command:
Writing a Unix-Like Shell for MS-DOS



by Allen Holub

This book and ready-to-use program demonstrate how to write a Unix-like shell for MS-DOS, with techniques applicable to most other programming languages as well. The book and disk include a detailed description and working version of the Shell, complete C source code, a thorough discussion of low-level MS-DOS interfacing, and significant examples of C programming at the system level.

Supported features include: read, aliases, history, redirection and pipes, Unix-like command syntax, MS-DOS-compatible prompt support, C-Shell-based shell scripts, and a Shell variable that expands to the contents of a file so a program can produce text that is used by Shell scripts. The Unix-like control flow includes: if/then/else; while; foreach; switch/case; break; and continue.

The ready-to-use program and all C source code are included on disk. For IBM PC and direct compatibles.

Book & Disk (MS-DOS) **Item #29-1** **$39.95**

\Util

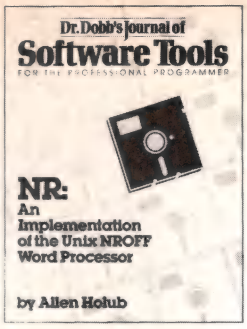
by Allen Holub

When used with the Shell described in **On Command**, this collection of utility programs and subroutines provides you with a fully functional subset of the Unix environment. Many of the utilities may also be used independently. You'll find executable versions of cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod.

The **\Util** package includes complete source code on disk and all programs (and most of the utility subroutines) are fully documented in a Unix-style manual.

Manual & Disk (MS-DOS) **Item #12-7** **$29.95**

NR:
A Implementation of the Unix NROFF Word Processor



by Allen Holub

NR is a text formatter written in C and compatible with Unix's NROFF. Complete source code is included in the NR package so that it can be easily customized to fit your needs. NR also includes an implementation of the -ms (manuscript) macro package and an in-depth description of how -ms works. NR does hyphenation and simple proportional spacing. It supports automatic table of contents and index generation, automatic footnotes and endnotes, italics, boldface, overstriking, underlining, and left and right margin adjustment. NR also contains:

- extensive macro and string capabilities
- number registers in various formats, including Roman and Arabic numerals, both spelled out and in outline form
- diversions and diversion traps (macros that are triggered automatically)
- input and output line traps

NR comes configured for any Diablo-compatible printer, as well as Hewlett Packard's ThinkJet, and LaserJet. It is easily configurable for most other printers. Both the ready-to-use program and full source code are included. For PC compatibles.

Manual & Disk (MS-DOS) **Item #33-X** **$29.95**

TO ORDER:
Mail in the attached postage paid order card or
CALL TOLL FREE 800-533-4372
(in CA 800-356-2002)

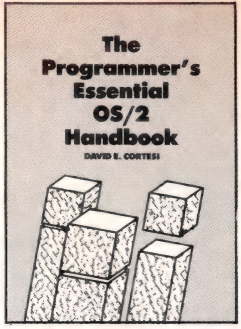
SPECIAL OFFER

Receive **On Command**, **\Util**, and **NR** together for only $85.95! You save 15% !

Unix-like Features Package **Item #167** **$85.95**

** Also available at your local bookstore*

The Programmer's Essential OS/2 Handbook



by David E. Cortesi

The Programmer's Essential OS/2 Handbook will let you harness the power of OS/2, without getting lost in its intricacies. This hands-on, working reference book is organized to guide you through the complex features of the new OS/2 system. You'll even find detailed technical information that is not included in the official OS/2 documentation. Multiple indices and a web of cross-referencing provide easy access to all OS/2 topic areas. Equal support for Pascal and C programmers is provided. You'll find:

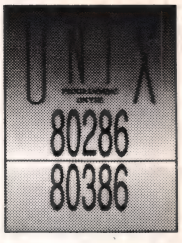
- an overview of OS/2 architecture and vocabulary, including references to where the book handles each topic in depth
- a look at the 80286 and a description of how the CPU processes data in real and protected mode
- an overview of linking, multiprogramming, file access, and device drivers
- in-depth discussions of important OS/2 topics, including dynamic linking; the message facility; the screen group; keyboard, mouse, and screen input, output and monitors; the queue; the semaphore; the thread; the process; storage allocation; segment swapping; and IOCTL usage
- detailed accounts of nearly 300 system calls, including DOS calls, keyboard calls, video calls, mouse calls, and device driver aids.

The Programmer's Essential OS/2 Handbook is a resource no programmer developing in the OS/2 environment can afford to be without.

Book & Disk (5-1/4" & 3" OS/2)	**Item #89-5**	**$39.95**
Book	**Item #82-8**	**$24.95**

Essential Operating System Information for Your Toolbox

Unix Programming on 80286/80386



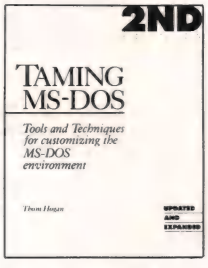
by Alan Diekman

Unix Programming on 80286/80386 provides experienced system programmers with an overview of time-saving Unix features, and an in-depth discussion of the relationship between Unix and DOS including many helpful techniques specific to programming under the Unix environment on a PC. The book also includes complete coverage of:

- the Unix program environment
- the Unix file system
- Unix shells
- basic Unix utilities
- C programming under Unix
- mass storage programs
- 80286 and 80386 architecture
- segment register programming
- Unix administration and documentation

Unix Programming on 80286/80386 contains many useful examples of the device drivers necessary to communicate with PC peripherals. It also includes useful information on how to set up device drivers for AT compatibles, such as cartridge tape drives and raster scan devices. Many examples of actual code are provided.

Book & Disk (Unix 5-1/4")	**Item #91-7**	**$39.95**
Book	**Item #83-6**	**$24.95**



Taming MS-DOS

2nd Edition

by Thom Hogan

A new and improved version of Hogan's popular first edition, **Taming MS-DOS Second Edition** has been updated to cover MS-DOS 3.3. You'll learn how to maximize your batch files with routines using redirection, filters and pipes, and routines that prevent accidental reformatting of your hard disk, redefine function keys, and locate files within subdirectories. Other batch files will implement an MS-DOS help wywtem, including help text files, a menu system that interprets keyboard input, and routine for quick redefinition of function keys.

You'll also learn how to create configurable AUTOEXE C.BAT files, and how to customize CONFIG.SYS and use ANSI.SYS to change the appearance of MS-DOS. **Taming MS-DOS Second Edition** includes ready-to-use BASIC programs that enhance MS-DOS. You can rename directories and disk volumes, change file attributes, check available RAM and disk memory, display a memory-resident clock, and assign MSDOS commands to ALT keys.

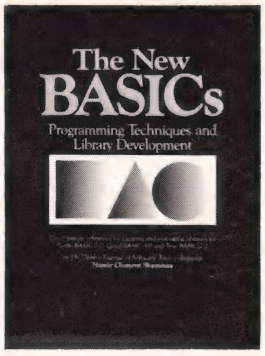
Updated and Expanded to cover MS-DOS 3.3 — The 2nd Edition is better than ever

The programs, including batch files and MS-DOS enhancements, are available on disk with full source code.

Book & Disk (MS-DOS)	**Item #92-5**	**$34.95**
Book	**Item #87-9**	**$19.95**

**** Also available at your local bookstore***

Programming Techniques in C, Forth, or BASIC —



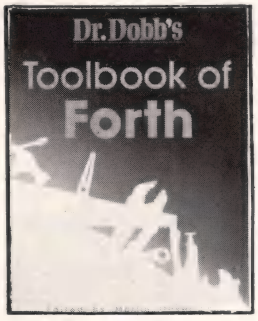
The New BASICs: Programming Techniques and Libraries

by Namir Clement Shammas

This book will orient the BASIC programmer to the syntax and programming features of the new BASICs, including Turbo BASIC, Quick BASIC, and True BASIC. You'll learn the details of implementing subroutines, functions and libraries to permit more structured coding. The **New BASICs** contains many programming examples and ready-to-use libraries, including libraries for string manipulation, extended string management, and sorting and searching. You'll also find a binary tree library, DOS files library, a library for Pascal-like sets, and much more.

Book & Disk (MS-DOS)	**Item #43-7**	**$39.95**
Book	**Item #37-2**	**$24.95**

TO ORDER:
Mail in the postage paid order card on the previous page or CALL TOLL FREE 800-533-4372 (in CA 800-356-2002)



Dr. Dobb's Toolbook of Forth

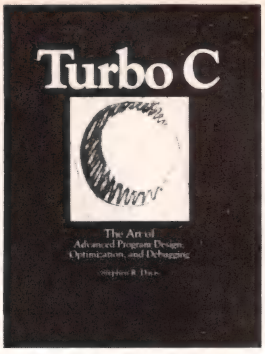
Edited by Marlin Ouverson

This comprehensive collection of useful Forth programs and tutorials contains expanded and revised versions of *Dr. Dobb's Journal of Software Tools'* best Forth articles, along with new Forth material. Contents include: mathematics in Forth, modifications/extensions, Forth programs, Forth – the language, and implementing Forth. You'll also find appendixes that will help you convert fig-Forth to Forth -83.

The screens in the book are available on disk as ASCII files.

Book & Disk (MS-DOS, Apple II, Macintosh, or CP/M: Osborne, 8: SS/SD)	**Item #57-7**	**$39.95**
Book	**Item #10-0**	**$22.95**

Turbo C: The Art of Advanced Program Design,Optimization and Debugging

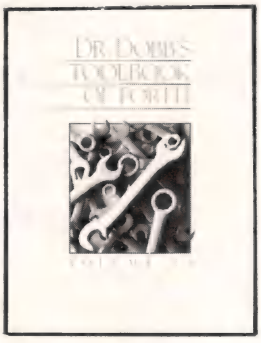


by Stephen R. Davis

Overflowing with example programs, this book fully describes the techniques necessary to skillfully program, optimize and debug in Turbo C. Every topic and Turbo C feature discussed is fully demonstrated in Turbo C source code examples. Advanced topics such as pointers; direct screen I/O; inline statements in Turbo C; and how to intercept and redirect BIOS calls are all covered in depth. The author further demonstrates these advanced topics by writing a RAM resident pop-up program in Turbo C. In addition the author fully outlines the differences between Unix C and Turbo C; the transition from Turbo Pascal to Turbo C; and the superset of K&R C features implemented in Turbo C and included in the proposed ANSI C standard.

Book & Disk (MS-DOS)	**Item #45-3**	**$39.95**
Book	**Item #38-0**	**$24.95**

Dr. Dobb's Toolbook of Forth, Volume II



by the editors of *Dr. Dobb's Journal of Software Tools*

This complete anthology of Forth programming techniques and development picks up where **Dr. Dobb's Toolbook of Forth** left off. You'll find the best articles of Forth from *Dr. Dobb's Journal of Software Tools*, along with the latest material from other Forth experts. Topics include: Forth philosophy and standards, programming windows, extended control stuctures, the design of a Forth target compiler, and more.

The screens in the book are available on disk as ASCII files.

Book & Disk (MS-DOS, Apple II, Macintosh, or CP/M: Osborne, 8" SS/DS)	**Item #51-8**	**$45.95**
Book	**Item #41-0**	**$29.95**

** Also available at your local bookstore*

Listing One *(Listing continued, text begins on page 106.)*

```
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ------------------------------ mouseUp ---------------------------
      on mouseUp
        -- make a noise
        play "boing" "e"

        -- hide this button
        hide button "Scouting Toolkit Icon"

        -- show the toolkit
        showScoutingToolkit
      end mouseUp


      ------------------------ showScoutingToolkit --------------------
      on showScoutingToolkit
        -- show the toolkit's buttons
        show button "Toggle Menubar"
        show button "Scouting Toolkit Window"
        show button "Scouting Toolkit About"
        show button "Scouting Toolkit™"
        show button "Toggle Message Box"
        show button "Toggle Tool Window"
        show button "Toggle Pattern Window"
        show button "Previous"
        show button "Next"
        show button "Back"
        show button "Home"
      end showScoutingToolkit

-------------------------------------------------------------

Button 14
   Description
      The button's name is "Duplicate Scouting Toolkit"
      Its Show name property is false
      Its Auto hilite property is true
      Its style is shadow
      It has an icon, ID# 30000
      Its rectangle is 42 units wide, 42 units high
      The button's located at card location (460,12)
      Its text properties are: System Font 12, Height 16, Align center
   Script
      ------------------------------ newButton ----------------------
      on newButton
        -- make sure we don't mess up the supply card
        if short name of this card is not "Scouting Toolkit" then

          -- save and set the user level
          get userLevel
          put it into entryUserLevel
          set userLevel to 5

          -- set the cursor to the watch
          set cursor to 4

          -- set the tool
          choose button tool

          -- for all the buttons in the set
          repeat with buttonNumber = 1 to 13

            -- remember where we are (the target card)
            push card

            -- hide screen commotion
            set lockScreen to true

            -- go to the supply card
            go to card "Scouting Toolkit" of stack "Scouting Toolkit™"

            -- select the button
            click at loc of button buttonNumber
```

(continued on next page)

Listing One *(Listing continued, text begins on page 106.)*

```
      -- copy the button
      doMenu "Copy Button"

      -- return to the target card
      pop card

      -- show screen commotion
      set lockScreen to false

      -- paste the button
      doMenu "Paste Button"

    end repeat

    -- copy the copyright notice
    copyCopyrightField

    -- get rid of this duplication button
    choose button tool
    click at loc of button "Duplicate Scouting Toolkit"
    doMenu "Cut Button"

    -- move two buttons
    drag from loc of button "Scouting Toolkit Icon" to loc of button "Back"
    hide button "Scouting Toolkit Icon"
    drag from loc of button "Delete Scouting Toolkit" to loc of button "Home"

    -- hide the toolkit
    choose browse tool
    click at loc of button "Scouting Toolkit Window"

    -- restore the user level
    set userLevel to entryUserLevel
  end if
end newButton

-------------------------- copyCopyrightField --------------------------
on copyCopyrightField
  -- save our location
  push card

  -- hide screen commotion
  set lockScreen to true

  -- go to the supply card
  go to card "Scouting Toolkit" of stack "Scouting Toolkit™"

  -- set tool
  choose field tool

  -- select the field
  show card field "Copyright"
  click at loc of card field "Copyright"

  -- copy the button
  doMenu "Copy Field"

  -- hide the field
  hide card field "Copyright"

  -- and get the field's contents
  get card field "Copyright"

  -- return to the target card
  pop card

  -- show screen commotion
  set lockScreen to false

  -- paste the field, fill it up, and hide it
  doMenu "Paste Field"
  put It into card field "Copyright"
  hide card field "Copyright"
end copyCopyrightField

------------------------------ mouseUp ------------------------------
on mouseUp
  -- make sure we're at the supply card
  if short name of this card is "Scouting Toolkit" then

    -- show the watch cursor
    set cursor to 4
```

```
        -- copy the button
        choose button tool
        click at loc of button "Duplicate Scouting Toolkit"
        doMenu "Copy Button"

        -- restore tool
        choose browse tool
      end if
    end mouseUp
```

```
Button 15
  Description
    The button's name is "Copyright"
    Its Show name property is false
    Its Auto hilite property is true
    Its style is transparent
    It has an icon, ID# 30000
    Its rectangle is 70 units wide, 13 units high
    The button's located at card location (0,52)
    Its text properties are: System Font 12, Height 16, Align center
  Script
    ------------------------------ mouseUp ------------------------------

    on mouseUp
      -- pass message on to the toolkit's About button
      send mouseUp to button "Scouting Toolkit About"
    end mouseUp
```

```
Card Field 1
  Description
    The field's name is "Copyright"
    It's hiddden
    Its Lock text property is true
    Its Show lines property is false
    Its Wide margins property is false
    Its style is rectangle
    Its rectangle is 286 units wide, 129 units high
    The field's located at card location (204,190)
    Its text properties are: Geneva 12, Height 16, Align center
  Contents
    "©1987 by Stan Krute's Camp Creek Institute.
    All rights reserved.
    Call or write for details:
    18617 Camp Creek Road
    Hornbrook, California96044
    (916) 475-3428"
  Script
    The field has no script
```

```
Card Field 2
  Description
    The field's name is "No Delete Here"
    It's hiddden
    Its Lock text property is true
    Its Show lines property is false
    Its Wide margins property is false
    Its style is opaque
    Its rectangle is 39 units wide, 39 units high
    The field's located at card location (37,293)
    Its text properties are: Geneva 9, Height 12, Align center
  Contents
    "Not Here,
    Though."
  Script
    The field has no script
```

End Listing

THE PROGRAMMER'S SHOP

helps save time, money, and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

MacFortran/020 by Absoft. Optimized FORTRAN-77 with VAX, 8X extensions, 68881 support, arrays memory or disk, complex math. Source debug, toolbox, C interfaces.
MAC 512, SE, II $449

AI Expert System Dev't

Arity Combination Package	PC	$ 979
CxPERT V3.0 - shell for C	MS	$ 349
Experteach II	PC	$ 339
Level 5 - formerly Insight 2	MS	$ 589
T.I.: PC Easy	PC	$ 435
Personal Consultant Plus	PC	$2589
Turbo Expert-Startup(400 rules)	PC	$ 119
Corporate (4000 rules)	PC	$ 339

AI-Languages

A.P.T. - Active Prolog Tutor	PC	$ 49
ARITY Prolog - Interpreter	PC	$ 229
PC Scheme LISP - by TI	PC	$ 79
TransLISP - learn fast	MS	$ 79
TURBO PROLOG by Borland	PC	$ 69
Turbo Prolog Toolbox	PC	$ 69

C Language-Compilers

AZTEC C86 - Commercial	PC	$ 499
C86 PLUS - by CI	MS	$ 359
Datalight Optimum-C	MS	$ 109
High C Optimizing Compiler	PC	Call
Instant-C - source, debug.	MS	$ 369
Instant-C/16M	PC	Call
Lattice C - from Lattice	MS	$ 259
Microsoft C 5.0 - Codeview		Special *
Microsoft Quick C		Special *
Rex-C/86-standalone ROM	MS	$ 695
Turbo C by Borland	PC	$ 67

C Language-Interpreters

C-terp by Gimpel - full K & R	MS	$ 219
C Trainer - by Catalytix	PC	$ 89
Interactive C by IMPACC Assoc.	PC	$ 189
Run/C Professional	MS	$ 145
Run/C Lite	MS	$ 79
Turbo C-terp	PC	$ 119

C Libraries-Files

BTree Source, no royalties	MS	$ 69
CBTREE-Source, no royalties	MS	$ 129
ctree by Faircom-no royalties	MS	$ 309
ctree w/ rtree	MS	$ 519
rtree - report generation	PC	$ 239
dB2C Files	MS	$ 259
dB2C Toolkit V2.0	MS	$ 249
dbVISTA - Object only		Special *
dBx - translator	MS	$ 299
w/source to library	MS	$ 419

FEATURES

CO/SESSION - Remotely access PC and peripherals, train or trouble-shoot from off-site, 2 users on one program. Session record/playback, file transfer, terminal emulation, keyboard and voice modes. PC $ 229

CLARION DBMS by Barrington Systems. Fast applications prototyping and development. Language, compiler, screen/report generators, editor, call other languages, read/write dBASE III + files. PC List: $ 695

***Mention "Special DD488" and get both a good price and FREE Software!**

FREE Catalogs and Guarantee

Whether you're searching for an obscure product no one seems to know about, or you just want to know which of 5 competitors makes the most well-regarded product, our catalogs make finding software easier.

- Comprehensive Product Listing, filled with over 1,000 products.
- Popular Programmer's Tools, containing the most-requested titles (over 300 in all) from each category.
- Dbase Programmer's Catalog with over 60 development tools.

We'll also help you select products with free advice or literature. Plus full guarantee on any recommended product. Mention "DD488."

Call to request a catalog or information today.

Our Services:

- International Sales Desk
- Compare Products
- Help find a Publisher
- Evaluation Literature FREE
- Programmer's Update
- Dealers Inquiry
- Newsletter
- Rush Order
- Over 700 products
- National Accounts Center

C Libraries-General

Blackstar C Function Library	PC	$ 99
C Tools Plus - V5.0	PC	$ 99
C Utilities by Essential	PC	$ 119
Entelekon C Function Library	PC	$ 119
Greenleaf C Sampler	PC	$ 69
Greenleaf Functions	PC	$ 129
LIGHT TOOLS by Blaise	PC	$ 69
Turbo C Tools by Blaise	PC	$ 99

C-Screens, Windows, Graphics

C Power Windows by Entelekon	PC	$ 129
C Worthy Interface Library	PC	$ 249
Curses by Aspen Scientific	PC	$ 109
dBASE Graphics for C	PC	$ 69
ESSENTIAL GRAPHICS-fast	PC	$ 185
FontWINDOW/PLUS	PC	$ 225
GraphiC - new color version	PC	$ 279
Greenleaf Data Windows	PC	$ 155
w/source	PC	$ 259
LightWINDOWS/C for Datalight	PC	$ 79
Panel/TC - for Turbo C	PC	$ 95
PC Forms-by Golden Software	MS	$ 109
ScreenStar - with source	PC	$ 169
Terminal Mapping System	PC	$ 279
TurboWINDOW/C-for Turbo C	PC	$ 75
View Manager - by Blaise	PC	$ 199
Vitamin C - source, menus	PC	$ 159
VC Screen - screen paint	PC	$ 79
Windows for C - fast	PC	Call
Windows for Data - validation	PC	Call
ZView - screen generator	MS	$ 139

DataBase & File Management

Advanced Revelation	PC	$ 779
CQL	PC	$ 359
DataFlex by Data Access	PC	$ 595
DataFlex multiuser	PC	$1049
Magic PC	PC	$ 169
Paradox - original	PC	$ 369
Paradox V2.0 List: $725	PC	$ 499

DBASE Language

Clipper compiler	PC	$ 399
dBASE III Plus	PC	$ 399
dBASE III LANPack	PC	$ 649
DBXL Interpreter	PC	$ 99
FoxBASE+ - V2.0	MS	$ 259
Quicksilver Diamond	PC	$ 369

DBASE Support

dAnalyst	PC	$ 219
dBASE Tools for C	PC	$ 65
dBRIEF with BRIEF-Auto-Indent, views structures	PC	Call

RECENT DISCOVERY

Foxbase PLUS by Fox Software. DBMS conforms to standard MAC interface. Up to 9 definable output windows, definable buttons, import/export hypercard stacks, MAC, Developer Kit, $339

DBASE Support cont.

dBC III by Lattice	MS	$169
dBC III Plus	PC	$509
dbug III	MS	$179
Dialect UI	PC	$ 45
dFLOW - flow charts	MS	$125
Documentor - dFlow superset	MS	$229
Genifer by Bytel-code generator	MS	$279
QuickCode III Plus	MS	$189
R&R Report Writer	MS	$139
Seek-It - Query-by-example	PC	$ 79
Silver Comm Library	MS	$139
Tom Rettig's Library	PC	$ 79
UI Programmer-user interfaces	PC	$249

Debuggers

Breakout - by Essential	PC	$ 89
CODESMITH - visual	PC	$ 99
C SPRITE - data structures	PC	$119
DSD87	PC	$ 79
Periscope I	PC	$275
Periscope II	PC	$139
Periscope II-X	PC	$105
Periscope III-10 MHZ Version	PC	$795
Pfix-86 Plus - by Phoenix	PC	$209
SoftProbe II - embedded systems	PC	$695

Editors for Programming

BRIEF Programmer's Editor	PC	Call
de - EMACS-style	PC	$ 65
EMACS by UniPress Source:	$895	$265
Epsilon - like EMACS	PC	$149
KEDIT - like XEDIT	PC	$ 99
ME Macro Editor - Source	PC	$ 79
MKS VI	MS	$ 65
PC/EDT - macros	PC	$229
PC/VI - by Custom Software	MS	$109
Personal REXX - V1.6	PC	$ 99
SPF/PC - Version 2.0	PC	$179
Vedit PLUS	MS	$129

Fortran & Supporting

ACS Time Series	MS	$399
Forlib+	PC	$ 55
Fortran Addenda	PC	$139
I/O Pro - includes No Limit	PC	$229
MACFortran by Microsoft		Special *
MS Fortran - 4.0, full 77'		Special *
PC-Fortran Tools - xref, pprint	PC	$165
RM/Fortran	MS	$399
Scientific Subroutines - Matrix	MS	$129

Multilanguage Support

BTRIEVE ISAM	MS	$185
BTRIEVE/N - multiuser	MS	$455
GSS Graphics Dev't Toolkit	PC	$375
Halo Development Package	MS	$389
HALO Graphics	PC	$209
Help/Control - on line help	PC	$ 99
Hoops 3D Graphics Library	PC	$469
Informix 4GL-application builder	PC	Call
Informix SQL - ANSI standard	PC	Call
Instant Programmer's Help	MS	$ 79

Note: Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/per normal item. All prices subject to change without notice.

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

CIRCLE NO. 161 ON READER SERVICE CARD

STRUCTURED PROGRAMMING

Listing One *(Text begins on page 118.)*

```
DEFINITION MODULE LineDwg;
  EXPORT QUALIFIED
    width, height, CharWidth, CharHeight, PaintMode, Px, Py,
    mode, dot, line, paint, copyArea, clear, Write, WriteString;

  TYPE PaintMode = (replace, add, invert, erase);

  VAR Px, Py     : INTEGER; (* Current coordinates of drawing pen *)
      mode       : PaintMode;  (* Current mode for paint and copy *)
      width      : INTEGER;    (* Width of picture area, read-only *)
      height     : INTEGER;  (* Height of picture area, read-only *)
      CharWidth  : INTEGER;              (* Width of a character *)
      CharHeight : INTEGER;             (* Height of a character *)

  PROCEDURE dot (c : CARDINAL; x, y : INTEGER);
        (* Place a dot at coordinate x, y in color c *)

  PROCEDURE line (d, n : CARDINAL);
        (* Draw a line of length n in direction d
           (angle = 45 * d degrees) *)

  PROCEDURE paint (c : CARDINAL; x, y, w, h : INTEGER);
        (* Paint the rectangular area at x, y of width w and
           height h in color c, where 0 = white, 1 = light gray,
           2 = dark gray, 3 = black *)
        (* Note: This proc is also called 'area' by Wirth *)

  PROCEDURE copyArea (sx, sy, dx, dy, dw, dh : INTEGER);
        (* Copy rectangular area at sx, sy into rectangle at dx, dy
           of width dw and height dh *)

  PROCEDURE clear;         (* Clear the screen *)
        (* Note: Also places display in EGA 640 x 350 color mode *)

  PROCEDURE Write (ch : CHAR);   (* Write ch at pen's position *)

  PROCEDURE WriteString (s : ARRAY OF CHAR);

END LineDwg.
```

End Listing One

Listing Two

```
IMPLEMENTATION MODULE LineDwg;

(* Implements LineDrawing module defined by N. Wirth in *)
(*   "Programming in Modula-2." This module assumes EGA *)
(*   monitor in 640 x 350 color graphics mode.          *)
(* Uses virtual coordinates, where origin is at lower   *)
(*   left corner of screen area, size is 800 x 600.     *)
(* Adapted by K. Porter for DDJ, April 1988 issue       *)
(* ---------------------------------------------------- *)

  FROM SYSTEM IMPORT REGISTERS, INT;
  FROM Strings IMPORT Length;

  CONST  VW  = 800.0;          (* virtual width of screen *)
         VH  = 600.0;                 (* virtual height *)
         RW  = 640.0;  (* real device width, EGA screen *)
         RH  = 350.0;                  (* device height *)
         EGA = 16;           (* EGA 640 x 350 color mode *)

  (* Variables local to this module *)
  VAR    reg    : REGISTERS;
         xf, yf : REAL;
         color  : INTEGER;

  (* --------------LOCAL PROCEDURES ------------------------ *)

  PROCEDURE writePixel (c, x, y : INTEGER);
                    (* write pixel of color c at device x, y *)

  BEGIN
    reg.AH := 12;
    CASE c OF            (* map color indicator to EGA palette *)
      0 : reg.AL := 15|                                (* white *)
      1 : reg.AL :=  7|                          (* light gray *)
      2 : reg.AL :=  8|                           (* dark gray *)
      3 : reg.AL :=  0                               (* black *)
    END;  (* of CASE *)
    reg.BX := 0;
    reg.CX := x;
    reg.DX := y;
    INT (16, reg);
    color := c;                          (* set prevailing color *)
  END writePixel;
  (* ------------------------ *)
```

```
PROCEDURE devX (x : INTEGER) : INTEGER;
                                (* translate x to device x *)

BEGIN
  RETURN TRUNC (xf * FLOAT (x));
END devX;
(* ------------------------ *)

PROCEDURE devY (y : INTEGER) : INTEGER;
                                (* translate y to device y *)

BEGIN
  RETURN TRUNC (RH - (yf * FLOAT (y)));
END devY;


(* ------------------ VISIBLE PROCEDURES -------------------- *)

PROCEDURE dot (c : CARDINAL; x, y : INTEGER);

      (* Place a dot of color c at coordinate x, y *)

BEGIN
  writePixel (c, devX (x), devY (y));
END dot;
(* ------------------------ *)

PROCEDURE line (d, n : CARDINAL);

      (* Draw a line of length n in direction d
        (angle = 45 * d degrees) *)

VAR  xdir, ydir : INTEGER;     (* x and y directions given d *)
     distance   : CARDINAL;

BEGIN
  CASE d OF
    0 : xdir :=  1; ydir :=  0|                    (* right *)
    1 : xdir :=  1; ydir :=  1|
    2 : xdir :=  0; ydir :=  1|                       (* up *)
    3 : xdir := -1; ydir :=  1|
    4 : xdir := -1; ydir :=  0|                    (* left *)
    5 : xdir := -1; ydir := -1|
    6 : xdir :=  0; ydir := -1|                    (* down *)
    7 : xdir :=  1; ydir := -1
  END;     (* of CASE *)
   FOR distance := 1 TO n DO
     Px := Px + xdir;                        (* advance the pen *)
     Py := Py + ydir;
     dot (color, Px, Py);          (* draw in prevailing color *)
   END;
END line;
(* ------------------------ *)
PROCEDURE paint (c : CARDINAL; x, y, w, h : INTEGER);

(* Paint the rectangular area at x, y of width w and
          height h in color c, where 0 = white,
          1 = light gray, 2 = dark gray, 3 = black *)

VAR  cy, prevY, dy : INTEGER;

BEGIN
  prevY := 0;
  color := c;                       (* set new prevailing color *)
  FOR cy := y TO y+h DO
    dy := devY (cy);                   (* get current device y *)
    IF dy <> prevY THEN              (* if new scan line, draw *)
      Px := x; Py := cy;
      line (0, w);
      prevY := dy;          (* remember where last line drawn *)
    END
  END
END paint;
(* ------------------------ *)

PROCEDURE copyArea (sx, sy, dx, dy, dw, dh : INTEGER);

      (* Copy rectangular area at sx, sy into rectangle
          at dx, dy of width dw and height dh *)

VAR  c, x, y, ix, iy, nx, ny, tx, ty : INTEGER;

BEGIN
  ix := devX (sx);    iy := devY (sy); (* source dev coords *)
  nx := devX (sx+dw); ny := devY (sy+dh);  (* ending coords *)
  tx := devX (dx);    ty := devY (dy);     (* target coords *)
  FOR y := ny TO iy DO                  (* go top to bottom *)
    FOR x := ix TO nx DO
      reg.AH := 13;                              (* read pixel *)
      reg.BX := 0;
```

(continued on next page)

STRUCTURED PROGRAMMING

Listing Two

(Listing continued, text begins on page 118.)

```
      reg.CX := x;
      reg.DX := y;
      INT (16, reg);              (* get pixel color into al *)
      reg.AH := 12;                 (* write pixel to dest *)
      reg.CX := tx + x - ix;
      reg.DX := ty + y - iy;
      INT (16, reg);
    END
  END
END copyArea;
(* ------------------------ *)

PROCEDURE clear;                              (* Clear the screen *)
            (* Also places display into EGA graphics mode *)

BEGIN
  reg.AH := 0;
  reg.AL := EGA;                               (* EGA 640 x 350 *)
  INT (16, reg);
  color := 0;                 (* reset default color to white *)
END clear;
(* ------------------------ *)

PROCEDURE Write (ch : CHAR);  (* Write ch at pen's position *)

VAR  cc, cr : INTEGER;                        (* Char col and row *)

BEGIN
  cc := devX (Px) DIV  8;         (* Derive char pos from pen *)
  cr := devY (Py) DIV 14;
  reg.AH := 2;                    (* Set text cursor position *)
  reg.BX := 0;
  reg.DX := (cr * 256) + cc;
  INT (16, reg);
  reg.AX := 2560 + ORD (ch);      (* Write char via ROM BIOS *)
  reg.BX := 7;                      (* Light gray text only *)
  reg.CX := 1;
  INT (16, reg);
  Px := Px + CharWidth;     (* advance by char virtual width *)
END Write;
(* ------------------------ *)

PROCEDURE WriteString (s : ARRAY OF CHAR);

VAR  i : CARDINAL;

BEGIN
    FOR i := 0 TO Length (s) DO
      Write (s[i]);
    END
  END WriteString;

  (* ---------------- INITIALIZATION ------------------------ *)

BEGIN
  Px          := TRUNC (VW / 2.0);     (* Virtual screen center *)
  Py          := TRUNC (VH / 2.0);
  mode        := replace;
  width       := TRUNC (VW);             (* Virtual screen size *)
  height      := TRUNC (VH);
  CharWidth   := 10;              (* Char sizes in virtual units *)
  CharHeight  := 24;
  xf          := RW / VW;                (* x translation factor *)
  yf          := RH / VH;                (* y translation factor *)
  color       := 0;                    (* white is default color *)
END LineDwg.
```

End Listing Two

Listing Three

```
MODULE Sierpin;

(* Based on the Sierpinski recursive model in N. Wirth's book *)
(*   "Programming in Modula-2."                                *)
(* Wirth's program has been modified slightly to accommodate  *)
(*   EGA graphics mode on the IBM PC.                          *)
(* Written by K. Porter for DDJ, April 1988 issue.           *)
(* ---------------------------------------------------------- *)

FROM InOut   IMPORT Read;
FROM LineDwg IMPORT width, height, Px, Py, clear, line;
FROM SYSTEM  IMPORT REGISTERS, INT;
```

(continued on page 92)

STRUCTURED PROGRAMMING

Listing Three

(Listing continued, text begins on page 118.)

```
CONST SquareSize = 512;

VAR  i, h, x0, y0 : CARDINAL;
     ch           : CHAR;
     reg          : REGISTERS;

PROCEDURE A (k : CARDINAL);
BEGIN
  IF k > 0 THEN
    A (k-1);   line (7, h);   B (k-1);   line (0, 2*h);
    D (k-1);   line (1, h);   A (k-1)
  END
END A;
(* -------------------------- *)

PROCEDURE B (k : CARDINAL);
BEGIN
  IF k > 0 THEN
    B (k-1);   line (5, h);   C (k-1);   line (6, 2*h);
    A (k-1);   line (7, h);   B (k-1)
  END
END B;
(* -------------------------- *)

PROCEDURE C (k : CARDINAL);
BEGIN
  IF k > 0 THEN
    C (k-1);   line (3, h);   D (k-1);   line (4, 2*h);
    B (k-1);   line (5, h);   C (k-1)
  END
END C;
(* -------------------------- *)

PROCEDURE D (k : CARDINAL);
BEGIN
  IF k > 0 THEN
    D (k-1);   line (1, h);   A (k-1);   line (2, 2*h);
    C (k-1);   line (3, h);   D (k-1)
  END
END D;
(* -------------------------- *)

BEGIN
  clear;
  i := 0;
  h := SquareSize DIV 4;
  x0 := CARDINAL (width) DIV 2;
  y0 := CARDINAL (height) DIV 2 + h;
  REPEAT
    i  := i + 1;
    x0 := x0 - h;
    h  := h DIV 2;
    y0 := y0 + h;
    Px := x0;
    Py := y0;
    A (i);   line (7, h);   B (i);   line (5, h);
    C (i);   line (3, h);   D (i);   line (1, h);
  UNTIL (i = 4);
  Read (ch);                              (* hold for keypress *)
  reg.AH := 0;
  reg.AL := 3;                (* restore 80 x 25 color text mode *)
  INT (16, reg);
END Sierpin.
```

End Listing Three

Listing Four

```
MODULE MathPlot;

(* Graphs the function y = cos x + sin 2x using LineDwg *)
(* K. Porter, DDJ, April 88                             *)

FROM MathLib0 IMPORT sin, cos, real, entier;
FROM LineDwg IMPORT line, dot, clear, WriteString, Px, Py;
FROM InOut IMPORT Read;
FROM SYSTEM IMPORT REGISTERS, INT;

VAR  x, sx, cx, y : REAL;
     xc           : INTEGER;
     ch           : CHAR;
     reg          : REGISTERS;

PROCEDURE yc (y : REAL) : INTEGER;
BEGIN
  RETURN entier (y * 100.0) + 300;
END yc;
```

```
BEGIN
  clear;
  Px :=   0; Py := 300; line (0, 800);          (* x axis *)
  Px :=   0; Py := yc (1.0);                     (* grids *)
  WriteString ("  1");
  dot (2, Px, Py);                          (* set dark gray *)
  line (0, 750);
  Px := 0; Py := yc (-1.0);
  WriteString (" -1"); line (0, 750);
  FOR xc := 125 TO 799 BY 126 DO        (* vertical grids *)
    Px := xc; Py := yc (2.0); line (6, 400);
  END;
  Px := 310; Py := 599;
  WriteString ("y = cos x + sin 2x ");

  FOR xc := 1 TO 799 DO
    x  := real (xc) / 80.0;                  (* x to radians *)
    sx := sin (2.0 * x);
    dot (2, xc, yc (sx));       (* plot sine in dark gray *)
    cx := cos (x);
    dot (1, xc, yc (cx));            (* cos in light gray *)
    y  := sx + cx;
    dot (0, xc, yc (y));        (* plot function in white *)
  END;
  Read (ch);
  reg.AH := 0; reg.AL := 3; INT (16, reg)  (* text mode *)
END MathPlot.
```

End Listing Four

Listing Five

```
MODULE Spiral;

(* Draws a spiral, then copies part of it to a window *)

FROM SYSTEM IMPORT REGISTERS, INT;
FROM LineDwg IMPORT Px, Py, line, copyArea, WriteString, clear;
FROM InOut IMPORT Read;

VAR  reg  : REGISTERS;
     ch   : CHAR;
     n, d : INTEGER;

BEGIN
  clear;
  Px := 250;
  FOR n := 0 TO 24 DO                          (* Draw the spiral *)
    d := n MOD 8;
    line (d, n * 5);
  END;
  Px := 548; Py := 198;             (* Outline the copy window *)
  line (0, 204);
  line (2, 204);
  line (4, 204);
  line (6, 204);
  Px := 630; Py := 440;
  WriteString ("Copy ");
  copyArea (200, 200, 550, 200, 200, 200);     (* Copy portion *)
  Read (ch);                                  (* Wait for keypress *)
  reg.AH := 0; reg.AL := 3; INT (16, reg);
END Spiral.
```

End Listings

Listing One *(Text begins on page 130.)*

```
SCR# 0

( Corrected source screens)     The following four screens contain
corrected source for the operators  GETDATA PUTDATA and PUTTEXT.
The original uncorrected versions are in the DDJ Forth column, Feb
1988.

Screen 5 defines GETDATA using
BEGIN ... DUP IF  DROP ... THEN WHILE ... REPEAT
instead of the preferred
BEGIN ... WHILE ... WHILE ... REPEAT THEN

SCR# 1

\ Read BLOCKed file as data file

: GETDATA ( a n - n2)
\ reads n bytes of data from input file into address, n < 64K
\ Returns n2 bytes not read ( ie beyond end of file ).
    ( calculate # of bytes to move < 64K : )  POSITION 2@
    BEGIN  2 PICK ( n ) DUP
      IF ( n ) >R  2DUP 1K UM/MOD  SWAP DROP  1+  1K UM*
         CAPACITY 2@ DMIN  2OVER D- 0= NOT OR  R> UMIN
      THEN  ?DUP
    WHILE  >R  2DUP 1K UM/MOD  BLOCK +  4 PICK R@ CMOVE
     R@ 0 D+  2SWAP  R> /STRING  2SWAP
    REPEAT  POSITION 2!  SWAP DROP ;

SCR# 2


\ Write BLOCKed file as data file

: PUTDATA ( a n)
\ writes n bytes of data to output file from address, n < 64K
   ( extend the file as needed : )
  DUP 0  POSITION 2@ D+  CAPACITY 2@  2OVER D-  DUP 0<
  IF  2DUP DABS EXTEND  2OVER CAPACITY 2!  THEN  2DROP 2DROP
  ( calculate # of bytes to move < 64K : )  POSITION 2@
  BEGIN  2 PICK ( n ) DUP
     IF ( n ) >R  2DUP 1K UM/MOD  SWAP DROP  1+  1K UM*
        CAPACITY 2@ DMIN  2OVER D- 0= NOT OR  R> UMIN
     THEN  ?DUP
  WHILE  >R  2DUP 1K UM/MOD  BLOCK +  4 PICK SWAP R@ CMOVE
     R@ 0 D+  2SWAP  R> /STRING  2SWAP   UPDATE
  REPEAT  POSITION 2!  2DROP ;

SCR# 3

 0 \ Read text file with #EOF
 1
 2 : GETTEXT ( a n - n2 f)   POSITION 2@   0 ( f ) >R
 3 \ reads n bytes of text from input file into address, n < 64K
 4 \ Returns n2 bytes not read ( ie end-of-line or beyond file)
 5 \ Returns true if #EOL terminates line; false otherwise.
 6    BEGIN  2DUP CAPACITY 2@ D<  3 PICK ( n ) AND
 7    WHILE  2DUP 1K UM/MOD  BLOCK + C@ ( get ch)
 8        DUP  #EOL =    DUP R> OR >R
 9        OVER #EOF = OR NOT  AND ?DUP
10    WHILE  >R  1 0 D+  2SWAP  R> 2 PICK C!  1 /STRING  2SWAP
11    REPEAT THEN   2DUP CAPACITY 2@ D<
12    IF  2DUP 1K UM/MOD  BLOCK + C@  #EOL = DUP D-  THEN
13    2DUP CAPACITY 2@ D<
14    IF  2DUP 1K UM/MOD  BLOCK + C@  #LF  = DUP D-  THEN
15    POSITION 2!  SWAP DROP  R> ;


SCR# 4

 0 \ Read text file without #EOF
 1
 2 : GETTEXT ( a n - n2 f)   POSITION 2@   0 ( f ) >R
 3 \ reads n bytes of text from input file into address, n < 64K
 4 \ Returns n2 bytes not read ( ie end-of-line or beyond file)
 5 \ Returns true if #EOL terminates line; false otherwise.
 6    BEGIN  2DUP CAPACITY 2@ D<  3 PICK ( n ) AND
 7    WHILE  2DUP 1K UM/MOD  BLOCK + C@ ( get ch)
 8       DUP  #EOL =    DUP R> OR >R
 9                      NOT  AND ?DUP
10    WHILE  >R  1 0 D+  2SWAP  R> 2 PICK C!  1 /STRING  2SWAP
11    REPEAT THEN   2DUP CAPACITY 2@ D<
12    IF  2DUP 1K UM/MOD  BLOCK + C@  #EOL = DUP D-  THEN
13    2DUP CAPACITY 2@ D<
14    IF  2DUP 1K UM/MOD  BLOCK + C@  #LF  = DUP D-  THEN
15    POSITION 2!  SWAP DROP  R> ;


SCR# 5
 0 \ Read text file with #EOF
 1
 2 : GETTEXT ( a n - n2 f)   POSITION 2@   0 ( f ) >R
 3 \ reads n bytes of text from input file into address, n < 64K
 4 \ Returns n2 bytes not read ( ie end-of-line or beyond file)
 5 \ Returns true if #EOL terminates line; false otherwise.
 6    BEGIN  2DUP CAPACITY 2@ D<  3 PICK ( n ) AND
 7    DUP IF  DROP  2DUP 1K UM/MOD  BLOCK + C@ ( get ch)
 8            DUP  #EOL =    DUP R> OR >R
 9            OVER #EOF = OR NOT  AND ?DUP  THEN
10    WHILE  >R  1 0 D+  2SWAP  R> 2 PICK C!  1 /STRING  2SWAP
11    REPEAT THEN   2DUP CAPACITY 2@ D<
12    IF  2DUP 1K UM/MOD  BLOCK + C@  #EOL = DUP D-  THEN
13    2DUP CAPACITY 2@ D<
14    IF  2DUP 1K UM/MOD  BLOCK + C@  #LF  = DUP D-  THEN
15    POSITION 2!  SWAP DROP  R> ;
```

End Listing

C CODE FOR THE PC

source code, of course

C Source Code

Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
CQL Query System (SQL retrievals plus windows)	$325
GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$295
Aspen Software PC Curses (System V compatible, extensive documentation)	$250
Vitamin C (MacWindows)	$200
Essential resident C (TSRify C programs, DOS shared libraries)	$165
Essential C Utility Library (400 useful C functions)	$160
Essential Communications Library (C functions for RS-232-based communication systems)	$160
Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
Greenleaf Functions (296 useful C functions, all DOS services)	$150
OS/88 (U**x-like O/S, many tools, cross-development from MS-DOS)	$150
Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
American Software Resident-C (TSRify C programs)	$130
CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
The Profiler (program execution profile tool)	$100
Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
Wendin O/S Construction Kit or PCNX, PCVMS O/S Shells	$95
QC88 C Compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
ME (programmer's editor with C-like macro language by Magma Software)	$75
WKS Library (C program interface to Lotus 1-2-3 program & files)	$65
Quincy (interactive C interpreter)	$60
EZ_ASM (assembly language macros bridging C and MASM)	$60
PTree (parse tree management)	$60
HELP (pop-up help system builder)	$50
Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
Heap Expander (dynamic memory manager for expanded memory)	$50
Make (macros, all languages, built-in rules)	$50
Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
Coder's Prolog (inference engine for use with C programs)	$45
C-Help (pop-up help for C programmers ... add your own notes)	$40
Biggerstaff's System Tools (multi-tasking window manager kit)	$40
CLIPS (rule-based expert system generator, Version 4.0)	$35
TELE Kernel (Ken Berry's multi-tasking kernel)	$30
TELE Windows (Ken Berry's window package)	$30
Clisp (Lisp interpreter with extensive internals documentation)	$30
Translate Rules to C (YACC-like function generator for rule-based systems)	$30
6-Pack of Editors (six public domain editors for use, study & hacking)	$30
ICON (string and list processing language, Version 6 and update)	$25
LEX (lexical analyzer generator)	$25
Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
AutoTrace (program tracer and memory trasher catcher)	$25
C Compiler Torture Test (checks a C compiler against K & R)	$20
Benchmark Package (C compiler, PC hardware, and Unix system)	$20
TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
A68 (68000 cross-assembler)	$20
List-Pac (C functions for lists, stacks, and queues)	$20
XLT Macro Processor (general purpose text translator)	$20

Data

WordCruncher (text retrieval & document analysis program)	$275
DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
Webster's Second Dictionary (234,932 words)	$60
U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12

Free shipping on prepaid orders **For delivery in Texas add 7%** **MasterCard/VISA**

MICRO/SYSTEMS

JOURNAL FOR THE PC SYSTEMS INTEGRATOR

SUBSCRIBE NOW AND SAVE OVER 47% OFF THE NEWSSTAND PRICE!

47% SAVINGS

Yes! I want to subscribe to
MicroSystems
JOURNAL for the PC Systems Integrator

and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

*Savings based on the full one-year newsstand rate of $47.70. All foreign countries please add $7 per year for surface mail; Canada & Mexico add $17 per year for airmail; other countries add $28 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

A PUBLICATION OF M&T PUBLISHING, INC. 212S

47% SAVINGS

Yes! I want to subscribe to
MicroSystems
JOURNAL for the PC Systems Integrator

and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

*Savings based on the full one-year newsstand rate of $47.70. All foreign countries please add $7 per year for surface mail; Canada & Mexico add $17 per year for airmail; other countries add $28 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

A PUBLICATION OF M&T PUBLISHING, INC. 212S

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro Systems
JOURNAL for the PC Systems Integrator

Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro Systems
JOURNAL for the PC Systems Integrator

Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

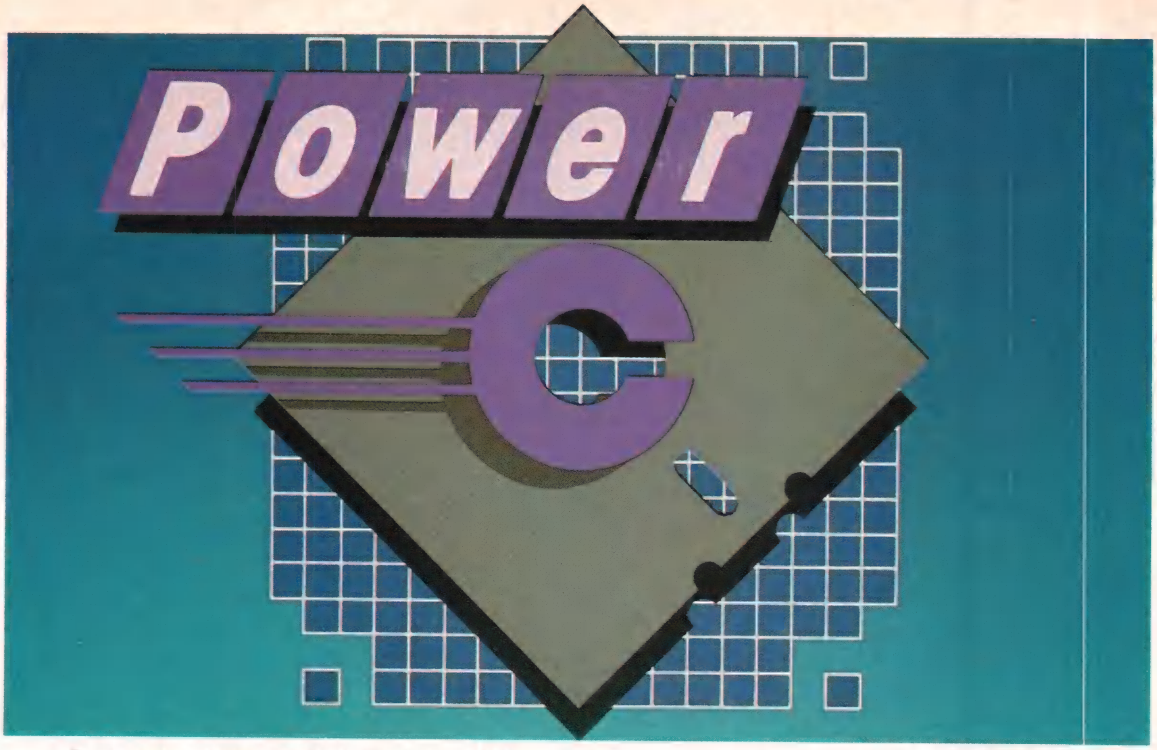
BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro Systems
JOURNAL for the PC Systems Integrator

Box 3713
Escondidio, CA 92025-9843



The $19.95 High-Performance C Compiler

Mix Software presents Power C . . . Our new cost-efficient alternative to high-priced C compilers. Now you can create high-performance programs without spending all your hard-earned money. But price isn't the only reason to choose Power C over the competition.

Compare the performance. Power C's integrated *Make* saves you time and effort by automatically managing your large programming projects. And with Power C, your programs can be as large as available memory. As for speed, the performance chart speaks for itself. Power C executes most of the benchmarks faster. And Power C creates smaller EXE files, out-performing the competition.

Performance Chart
(execution times in seconds)

	Power C	MS C	Turbo C
1) fib*	23.8	47.0	26.4
2) sieve*	27.6	40.2	25.5
3) tdbl*	3.5	9.0	9.6
4) diskio*	13.5	14.2	14.3
5) report**	11.0	86.3	60.7
6) drystone**	36.6	38.2	31.8
Compile/Link	73.9	187.6	81.4
EXE File Size	25120	29008	27184

Benchmarks from Dr. Dobb's Journal* & Computer Language**. First four programs test 1) function calling, 2) loops/integer math 3) floating point math, & 4) disk I/O. Programs 5 & 6 simulate typical applications. Tests compiled from command line using Make supplied with each compiler. Tests run on 8 MHz AT with medium model of Power C 1.0, MS (Microsoft) C 4.0, & Turbo C 1.0.

CIRCLE NO. 176 ON READER SERVICE CARD

Compare the functions. With over 400 functions, the Power C library is vastly superior. Our library is a superset of Microsoft C and Turbo C. Plus, we've added an extensive set of graphics functions for drawing lines, boxes circles, pie charts, and more.

Compare the portability. Power C supports the latest features of the proposed ANSI C standard. Plus, Power C is compatible with both Microsoft C and Turbo C. All of which makes it easier to move programs to and from Power C.

Compare the documentation. Our competition assumes that you're already a C wizard. We don't. Power C includes a step-by-step tutorial and sample programs with every function. With our complete documentation, programming in C couldn't be easier.

Power C is factors less expensive. And the source code to our function library is available at a fraction of their price.

Price Chart

C Compiler	Power C $19.95	MS C $450.00	Turbo C $99.95
Library Source Code Option	$10.00	N/A	$150.00
Total Cost with Source	$29.95	N/A	$249.95

MIX software

Technical Specifications

Power C includes: Power C compiler with integrated Make, Power C linker, Power C Libraries, Power C book, and support for...

- ✓ ANSI standard
- ✓ IEEE floating point
- ✓ 8087/80287 coprocessor
- ✓ auto-sensing of 8087/80287
- ✓ automatic register variables
- ✓ mixed model (near & far pointers)
- ✓ CGA, EGA, & Hercules graphics

Options are...

- ✓ Library source code
- ✓ BCD business math

Order Power C now by calling our toll free number or mail the coupon to Mix Software, 1132 Commerce Drive, Richardson, TX 75081.

1-800-523-9520

For technical support and for orders inside Texas call: 1-214-783-6001

Minimum System Requirements:
MSDOS or PCDOS 2.0 or later, 256K memory, 2 floppy drives or hard drive recommended, Runs on IBM PC, XT, AT, and compatibles, and IBM PS/2 model 25, 30, 50, 60, or 80.

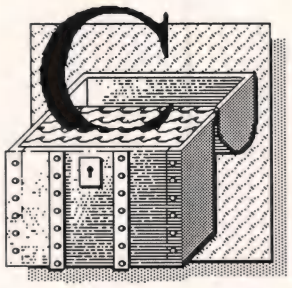
60 day money back guarantee

Name _____
Street _____
City _____
State _____ Zip _____
Telephone _____
Paying by: ☐ Check ☐ Money Order
☐ MC/Visa# _____ Exp _____
Computer Name _____ Disk Size ☐ 5¼" ☐ 3½"
Product(s) (Not Copy Protected)
☐ Power C ($19.95) $ _____
☐ Library Source Code ($10) (includes an assembler) $ _____
☐ BCD Business Math ($10) $ _____
Texas Residents add 8% Sales Tax $ _____
Add Shipping ($5 USA - $20 Foreign)
Total amount of your order $ _____

Power C is a trademark of Mix Software.
Microsoft C is a registered trademark of Microsoft Corporation.
Turbo C is a registered trademark of Borland International.

C CHEST

Formatted Print Functions: The Innards



A few months ago (August 1987), I presented the basic theory behind writing a subroutine with a variable number of arguments. This month I'm going to apply the theory by presenting a version of *printf( )*. "Why rewrite *printf( )*," you may well ask, "when you already have a version in the library?" There are two main reasons: size and features.

My version is an integer-only version of the subroutine. Because the normal *printf( )* can handle floating-point conversions, the linker will call the entire floating-point library into your program, even if it doesn't use floating point. The second size-related problem is implementation dependent. Microsoft's *printf( )* uses up an inordinate amount of stack space (which causes several problems in some applications, such as programs that use the multitasking kernel presented in this column in December and January). I suspect that it builds the entire output string in memory before sending it to the output device. My own version uses a different approach, which I'll discuss later.

The other issue is features. I wanted a %*b* (for binary) conversion that would write out a number in base 2, and I wanted some mechanism for centering an object in a field (as compared the default right justification or the left justification available with a minus-sign modifier). The various conversions supported by my *printf( )* are detailed in Table 1, page 100.

by Allen Holub

The %*p* conversion prints a 32-bit *far* pointer in a *SEG:OFFSET* form (use %*x* to print a 16-bit *near* pointer). To use it in a small-model program, you'll have to use a cast:

```
printf("%p", (void far *)ptr );
```

Note that it's impossible to get zero fill in the offset part of a %*p* conversion. That is 1234:0001 is always printed as 1234:1. You can use %*0p* to pad out the segment portion, however.

Implementation

Printf( ), *fprintf( )*, and *sprintf( )* all use a single workhorse function to do the work. This function, called *idoprnt( )*, is called with four arguments: a pointer to an output function, an argument to pass to that function in addition to the output character, the format-string pointer, and a pointer to the position on the stack of the rest of the arguments. Examples 1 and 2, below, show how it's used.

Figure 1, page 104, shows a picture of the stack after a call to:

```
sprintf( buf, "%d %lx", 1, (long)2 );
```

(I've chosen the *sprintf( )* call because it's the most complicated.) Arguments are pushed on the stack in reverse order, so the rightmost two arguments are pushed first: a *long int* holding the number *2* and a normal *int* holding the number *1*. A pointer to the format string is pushed next, followed by a pointer to the output buffer. *Sprintf( )* calls *idoprnt( )*, pushing a pointer to the first argument, a duplicate of the format-string pointer (if you don't understand this, go back and read the August C Chest), the address of the buffer pointer and a pointer to the output routine *putstr( )*, which will be called to output every character. *Putstr( )* is passed the same pointer to the buffer pointer that was passed to *idoprnt( )*, along with the character to output.

So, *putstr( )* outputs a character by putting it into ***p*. A glance at Figure 1 shows you that two levels of dereferencing gets you to a character in the buffer itself. The routine then increments **p*—that is, it increments the *buf* variable that's in *sprintf( )*'s stack frame.

Idoprnt( ) is presented in Listing One, page 68. I've done a few questionable things here—at least from the point of view of structured programming—primarily for speed reasons. If you run the profiler on most C programs, you'll find that a large

```
#include <stdio.h>
#include <stdarg.h>

printf( fmt, ... )
char    *fmt;
{
    extern int  fputc();
    va_list     args;

    va_start(args, fmt);
    idoprnt( fputc, stdout, fmt, args );
}

fprintf( stream, fmt, ... )
FILE    *stream;
char    *fmt;
{
    extern int  fputc();
    va_list     args;

    va_start(args, fmt);
    idoprnt( fputc, stream, fmt, args );
}
```

Example 1: Printf *and* fprintf

```
#include <stdarg.h>

putstr(c, p)
int     c;
char    **p;
{
    **p = c ;
    (*p)++  ;
}

int sprintf( buf, fmt, ... )
FILE    *stream;
char    *fmt;
{
    extern int  fputc();
    va_list     args;

    char *start = buf;
    va_start(args, fmt);
    idoprnt( fputc, &buf, fmt, args );
    *buf = '\e0' ;
    return buf - start;
}
```

Example 2: Sprintf

Blaise puts the Accent on C with

C TOOLS PLUS/5.0™

Now, just $129 and supports Microsoft C 5.0 and QuickC

Enhance your Microsoft C programming environment with C TOOLS PLUS/5.0™— a new, quintessential library of C functions. C TOOLS PLUS/5.0 from Blaise Computing Inc. puts a prime accent on quickly building professional applications using the full power of Microsoft C Version 5.0 and QuickC. Now you can concentrate on program creativity by having full control over DOS, menus, interrupt service routines, memory resident programs, printer and keyboard control, and more!

C TOOLS PLUS/5.0 prebuilt libraries are ready to use with either QuickC or the Microsoft C Version 5.0 command line environment. Complete documented source code is included so that you can study and adapt it to your specific needs. Blaise Computing's attention to detail, like the use of full function prototyping, cleanly organized header files, and a comprehensive, fully-indexed manual, makes C TOOLS PLUS/5.0 the choice for experienced developers as well as newcomers to C.

Continuous refinement of Blaise Computing's library products has produced a collection of tools that are unsurpassed for reliability, functionality and ease of use. Built upon the widely acclaimed C TOOLS PLUS, C TOOLS PLUS/5.0 includes such highly-developed features as:

◆WINDOWS
- Stackable, removable.
- Optional borders, cursor memory.
- Accept user input, formatted output.
- "printf" window-oriented output. **NEW!**

◆INTERRUPT SERVICE ROUTINES
- Capture DOS critical errors and keystrokes.
- Install hardware interrupt handlers.

◆RESIDENT SOFTWARE SUPPORT
- Install, detect and remove memory resident programs.

◆MENUS
- Horizontal and pulldown. **NEW!**
- Lotus-style support. **NEW!**

◆INTERVENTION CODE
- Schedule C functions at specified times, intervals or with a "hot key." **NEW!**
- Take full advantage of DOS, even from memory resident programs. **NEW!**

◆FAST DIRECT VIDEO ACCESS
- All monitors, even EGA 43-line mode.

◆PRINTER CONTROL
- Access BIOS print functions. **NEW!**
- Control the DOS PRINT utility. **NEW!**

◆UTILITIES AND MACROS
- Take advantage of DOS file structure.
- Manipulate data types, far & near pointers. **NEW!**
- Access any memory areas with fast "peek" and "poke" macros. **NEW!**

C TOOLS PLUS/5.0 supports the Microsoft C Version 5.0 and QuickC compilers, requires DOS 2.00 or later and is just **$129.00.**

C ASYNCH MANAGER™ Version 2.0 IMPROVED!

C ASYNCH MANAGER is a library of functions designed to help you incorporate asynchronous communication capabilities into your application programs. Version 2.0 has been rewritten especially for Microsoft C Version 5.0 and Borland's Turbo C. Simultaneous buffered input and output to both COM ports at speeds up to 9600 baud, XON/XOFF protocol, modem control and XMODEM file transfer are among the many features supported and is priced at just **$175.00.**

Blaise Computing Inc. has a full line of support products for both Pascal and C. Call today for your free information packet.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

THE BLAISE MENU

Turbo C TOOLS *$129.00*
Windows; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. For Turbo C.

Turbo POWER SCREEN *$129.00*
NEW! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. Now for Turbo Pascal 4.0, soon for C and BASIC.

Turbo POWER TOOLS PLUS *$129.00*
NEW VERSION! Now supports Turbo Pascal 4.0. Screen, window and menu management including EGA support; DOS memory control; ISRs; scheduled intervention code; and much more.

Turbo ASYNCH PLUS *$129.00*
NEW VERSION! Now supports Turbo Pascal 4.0. Interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 19.2K baud; modem control and XMODEM file transfer.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM. For MS-Pascal.

KeyPilot *$49.95*
"Super-batch" program. Create batch files which can invoke programs and provide input to them; run any program unattended; create demonstration programs; analyze keyboard usage.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

RUNOFF *$49.95*
Text formatter for all programmers; flexible printer control; user-defined variables; index generation; general macro facility. Crafted in Turbo Pascal.

TO ORDER CALL TOLL FREE
800-333-8087

TELEX NUMBER - 338139

Yes! Send me the prime accent!
Enclosed is $_____ for _____ copies of _____.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: _____ Phone: (_____) _____
Address: _____ State: _____ Zip: _____
City: _____ Exp. Date: _____
VISA or MC#: _____

CIRCLE NO. 177 ON READER SERVICE CARD

(continued from page 98)

percentage of the program's execution time is spent in *printf( )*. Consequently, it's worthwhile to speed things up a little, even if the code suffers a bit as a consequence. Also note that I've used the ANSI (not Unix) variable-argument conventions, all defined in <stdarg.h>, included on line 1.

The <dos.h> file included on line 2 is supplied by Microsoft and contains *#defines* for the *FP_OFF* and *FP_SEG* macros, which are used to extract the offset and segment portions of a *far* pointer. There really isn't a portable way to do this, but one possibility is:

```
#define FP_OFF(fp) ((unsigned)(fp))
#define FP_SEG(fp) ((unsigned)
        ((unsigned long)(fp) >> 16))
```

Note that I'm counting on truncating a 32-bit pointer to 16 bits in the cast to *unsigned*. You have to cast the pointer to a *long int* in order to shift it.

There are three macros of interest on lines 57–68 of Listing One. *PAD* outputs *fw* characters (*fc*), using the indicated output-function pointer (*out*) and passing that function an additional argument (*op*). To send five spaces to *stdout*, use:

```
int fp = 5;
PAD( fp, ' ', fputc, stdout );
```

The first argument must be a variable reference (not a constant). Note that an undesirable side effect of *PAD* sets *fp* to 0 when it terminates.

TOINT(p,x) converts the integer represented by the string *p* to an *int* and puts the result into *x*. It modifies *p* to point past the rightmost digit. *INTMASK* is a portable way to mask off the bottom *int*-size number of bits in a *long*. That is, given a 16-bit *int* and a 32-bit *long*, I want to mask out the bottom 16 bits. I can't say:

```
long x;
x &= 0xffff ;
```

because *0xffff* is treated by the compiler as an *int*. Moreover, it's negative. The &= will cause an implicit type conversion from *int* to *long*, performing a sign extension as part of the conversion (*0xffff* will be converted to *0xffffffff*), and the *AND* operation will have no effect. To get around this problem, you have to cast the *0xffff* to *unsigned* to defeat the sign extension:

```
x &= (unsigned)0xffff ;
```

The next problem is that *0xffff* assumes that an *int* is 16 bits. It's better to say:

```
void printf( char *fmt, . . . );
Basic conversions:
%d %ld   int decimal,long decimal
%u       unsigned int (only, no longs)
%s       string
%x %lx   int hex, long hex
%o %lo   int octal, long octal
%b %lb   int binary, long binary
%p       far pointer (in hex XXXX:XXXX)

Supported modifiers (may be combined):
%0x      Zero fill to left of number or string
%–x      Left justify number in field
%¦5      Center number in field
%10x     Print number in a 10-character-wide field
%*x      Get field width from the next argument
%10.5s   Strings only: print at most 5 characters from string in a 10-character-wide
         field. If either number is replaced by a *, get the corresponding width from
         the next argument. The following are equivalent:
             printf("%10.5s", tr );
             printf("%*.*s" 10, 5, str );
```

Table 1: Printf( ) *conversion summary*

Fast database development system with SQL-based db_QUERY and Lotus 123 interface. . . ™

C PROGRAMMERS- We asked what you wanted in a database development system and we built it!

db_VISTA III™ is the database development system for programmers who want powerful, high performance DBMS capabilities ... and in any environment. Based on the network database model and the B-tree indexing method, db_VISTA III gives you the most powerful and efficient system for data organization and access. From simple file management to complex database structures with millions of records. db_VISTA III runs on most computers and operating systems like MS-DOS, UNIX, VAX/VMS and OS/2. It's written in C and the complete source code is available, so your application performance and portability are guaranteed! With db_VISTA III you can build applications for single-user microcomputers to multi-user LANs, up to minis and even mainframes.

FAST • PORTABLE • ROYALTY-FREE

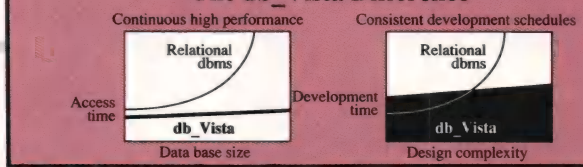
PROFESSIONAL SERVICES: In addition to 60 days of FREE technical support, we offer complete services to get your development project going and keep it on track:

Training Classes • Extended Support • Applications Development & C Programming Services • Consulting • Database Design & Optimization • Product Modification

We're committed to making your database project a success!

HOW TO ORDER: Call us; we'll help determine your needs and get you started. Add components as you need them. Ask about the new Lotus interface. . .Call today!

The db_Vista Difference



The db_VISTA III™ Database Development System

1 db_VISTA™: The High Performance DBMS
The major features include:
- Multi-user support for LANs and multi-user computers.
- Multiple database access.
- File and record locking.
- Automatic database recovery.
- Transaction processing and logging.
- Timestamping.
- Database consistency check utility.
- Fast access methods based on the network database model and B-tree indexing.
- An easy-to-use interactive database access utility.
- File transfer utilities for importing/exporting ASCII text and dBASE II/III files.
- A Database Definition Language patterned after C.
- Virtual memory disk caching for fast database access.
- A runtime library of over 100 functions.

2 db_QUERY:™ The SQL-based Query.
- Provides relational view of db_VISTA applications.
- Structured Query Language
- C linkable.
- Predefine query procedures or run ad-hoc queries "on the fly".

3 db_REVISE™: The Database Restructure Program.
- Redesign your database easily.
- Converts all existing data to revised design.

4 WKS Library for Lotus 123.
- C-linkable interface to Lotus files.

- **Operating systems:** MS-DOS, UNIX V, XENIX, VMS, OS/2.
- **C Compilers:** Lattice, Microsoft, IBM, Aztec, Computer Innovations, Turbo C, XENIX, and UNIX.
- **LAN systems:** LifeNet, NetWare, PC Network, 3Com, SCO XENIX-NET, other NETBIOS compatible MS-DOS networks.

All components feature royalty-free run-time distribution, source code availability and our commitment to customer service. That's why corporations like ARCO, AT&T, Hewlett-Packard, IBM, Northwestern Mutual Life, UNISYS and others use our products.

db_VISTA III™ Database Development System

db_VISTA III™	$595 - 3960
db_QUERY™	$595 - 3960
db_REVISE™	$595 - 3960
db_VISTA™ File Manager	Starts at $195
WKS Library for Lotus 123	Starts at $195

When high quality data base applications with outstanding performance are important to your company's success:

CALL 1-800-db-RAIMA

(that's 1-800-327-2462)

In the UK call Systemstar Ltd. 0992-500919

RAIMA™ CORPORATION

3055 112th Avenue N.E., Bellevue, WA 98004 **(206) 828-4636**
Telex: 6503018237MCIUW FAX: (206) 828-3131

DD48

Flotsam and Jetsam

Lvalues and Rvalues

One of the most frustrating error messages to beginning C programmers is "lvalue required." An understanding of lvalues and rvalues not only gets rid of the error message, but it can also help you understand complex expression evaluations.

First some etymology. In the expression:

```
x = y;
```

x is the lvalue (it's to the left of the equal sign) and *y* is the rvalue (because it's on the right). Lvalues are single variables that you have actually declared, or to be more accurate, the lvalue is the address of a single variable that you've declared. For example, an archetypal (but stupid) compiler, when given this input:

```
int x, *p, a[10];
x = 1;
p = &x;
*p = 1;
a[1] = 2;
```

will generate the following pseudo-8086 code:

```
mov __x,1

lea t0,__x
mov __p,t0

mov bp,__p
mov [bp],1

lea bp,__a
add bp,2 ; sizeof(int)
mov [bp],2
```

Here, *x*, *p*, **p*, and *a[1]* are all lvalues because all evaluate to addresses of single objects. In assembly language, *[bp]* (but not *bp* without the indirection) and the actual labels (*__p* and *__a*) are lvalues. An array name without the brackets doesn't form an lvalue because it doesn't evaluate to a single object. In the earlier code, a *bp* (without the brackets) is an rvalue. That is, it's a temporary variable that the compiler uses on the way to doing something else.

As another example of rvalues, the expression: *x = a + (b + c);* generates code like the following:

```
mov t0,__a
mov t1,__b
mov t3,__c
add t1,t3
add t0,t1
mov __x,t0
```

The contents of the variables are put into rvalues before they're evaluated, and the actual evaluation is done on the rvalues *t0*, *t1*, and *t3*, not on the real variables. (An optimizing compiler would clean this up, of course, but it's useful to look at the way that the compiler is actually thinking.)

To understand the lvalue-required problem, consider the code generated by *i=a++*:

```
mov t0,__a
add __a,1
mov __i,t0
```

and *i=++a*:

```
add __a,1
mov t0,__a
mov __i,t0
```

(Again, an optimizer would clean things up.) In both cases the *add* instruction affects the lvalue (*__a*), but the entire expression generates an rvalue (*t0*) that contains the result of the ++ operation. It's the rvalue that's used in the evaluation of the rest of the expression (the move to *__i*).

Now consider an (illegal) statement, such as *(a+b)++*. The compiler will try to do the following:

```
mov t0,__a ; (a+b)
mov t1,__b
add t0,t1

mov t1,t0 ; ++
add t0,1
```

There are two things to notice here. First of all, the subexpression *(a+b)* generates an rvalue—a temporary variable that holds the result of the addition. Next, the last two lines aren't doing anything useful. They're just flailing around manipulating temporary variables that are never used. That's why ++ requires an lvalue, because applying it to an rvalue or anonymous temporary creates meaningless code.

As another example of how a knowledge of rvalues is useful, consider the following complicated expression, which uses *argv* as pictured in Figure 2, below (the numbers are arbitrary addresses):

```
x = * ++ * ++ argv;
```

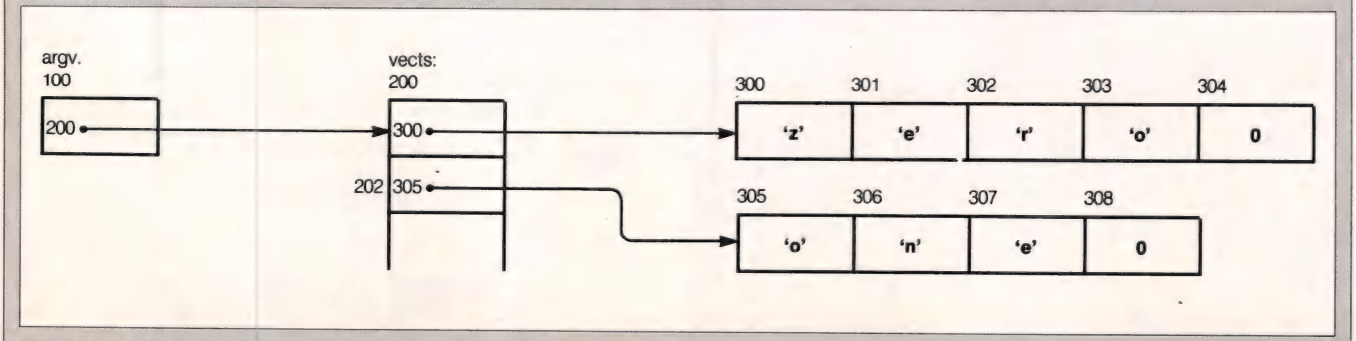


Figure 2: Argv

This expression copies the *'n'* (in "one") into *x*. To see how, think about rvalues and temporary variables. The compiler evaluates the expression from the inside out, starting at the name. The order of evaluation is determined by the order-of-precedence chart. Because * and ++ are at the same level but associate left to right, you get:

```
(x = (* (++ (* (++ argv)))))
```

The assignment is done last only because it's of lower precedence than either * or ++. The innermost ++*argv* generates the following code:

```
add   __argv,2   ; sizeof(char*)
mov   t0,__argv  ; t0 = 200 + 2 = = 202
```

so the ++ (the add instruction) was applied to a legitimate lvalue (*argv*) and generated an rvalue (*t0*) that contained *argv* after the increment. That is, given the addresses shown in the figure, the code modifies arg from 200 to 202 and then moves the 202 to *t0*. Now the compiler moves out a notch and sees the star. It applies that star to the rvalue that resulted from the previous subexpression, yielding:

```
mov   bp,t0      ; bp = 202
mov   t1,[bp]    ; t1 = 305
```

The star effectively changes the rvalue back into an lvalue (in *t1*) because *t1* holds the address of a single declared object—that is, it holds the address of *vects[1]* (202). You can tell it's an lvalue because of the brackets.

The compiler now moves out a notch and finds the second ++. Because *t1* is an lvalue, everything's OK and the following is generated:

```
add   t1,1       ; sizeof(char)
mov   t2,t1      ; t2 = 306 ;
```

Now it finds the next star and generates:

```
mov   bp,t2                ; bp = 306
mov   t3,BYTE PTR [bp]     ; t3 = 'n'
```

Finally, seeing the equal sign, it generates:

```
mov __x,t3.
```

There are two important things to notice here. First, the compiler is stupid. It always generates the same type of code for the same operator—only the names have been changed to protect the innocent. That is, a ++ preincrement always generates code having the following form:

```
add lvalue,sizeof(object)
mov temporary,lvalue
```

Similarly, a star always generates:

```
mov bp,object
mov temporary,[bp]
```

Second, the compiler always applies an operator to the temporary (or temporaries in the case of binary operators) that resulted from evaluating the previous subexpression. That is, it evaluates all expressions, no matter how complex, one operator at a time, and that operator is always applied to the temporary variable (read rvalue) that resulted from the previous evaluation. If you keep this fact firmly in mind, you can interpret any expression, no matter how tortuous it seems. —*A.H.*

```
x &= (unsigned)(~0);
```

which makes no assumptions about word size.

Idoprnt( ) itself begins on line 72 (page xx). Nonconversion characters are printed by the *for* loop on line 92 and output-routine call on line 96. The *else* clause comprises the remainder of the subroutine (it starts on line 100 and extends to line 259). The % conversions are all done in this *else* clause. The various modifiers are extracted on lines 115–142, and the *switch* on lines 152–207 controls the actual conversion.

I've used a *goto* on lines 165–169 to avoid an unnecessary subroutine call. The only *goto*-less way of doing the same thing (avoiding the subroutine call) that I could think of is:

```
base = 0;
...

case 'u': base = (-10 - 16 );
case 'x': base += ( 16 - 10 );
case 'd': base += ( 10 - 8 );
case 'o': base += ( 8 - 2 );
case 'b': base += ( 2 );
```

but that's sick. (That's a technical term.) To see what's going on, note that:

```
(-10-16)+(16-10)+(10-8)+(8-2)
+(2)                          ==-10
(16-10)+(10-8)+(8-2)+(2)      ==16
(10-8)+(8-2)+(2)              ==10
(8-2)+(2)                     ==  8
+(2)                          ==  2
```

You can see it better if you shuffle things around:

```
-10 +(16-16)+(10-10)+(8-8)+( 2-2)
                              ==-10
```

The multiple adds are both slow and abstruse, however, so a *goto* seems a better choice.

The masks on lines 186 and 190 defeat sign extension on nondecimal *int-size* numbers and *unsigned ints*. You can't allow sign extension on a hex conversion because it would add leading Fs to the number. If the number is negative and zero fill is active, the minus sign is printed on line 200. I do it here to avoid things such as 000–2 instead of –0002. The actual conversion is done by the *ltos( )* call on line 205, which I'll discuss shortly.

The code on lines 214 to 225 both '\0' terminates the string and figures the length. In the case of a %*s* conversion, *bp* will hold the same pointer that was passed into the original *printf( )* call. Because the string is already terminated, all you need is the length, extracted with a *strlen( )* call on line 222. If you did

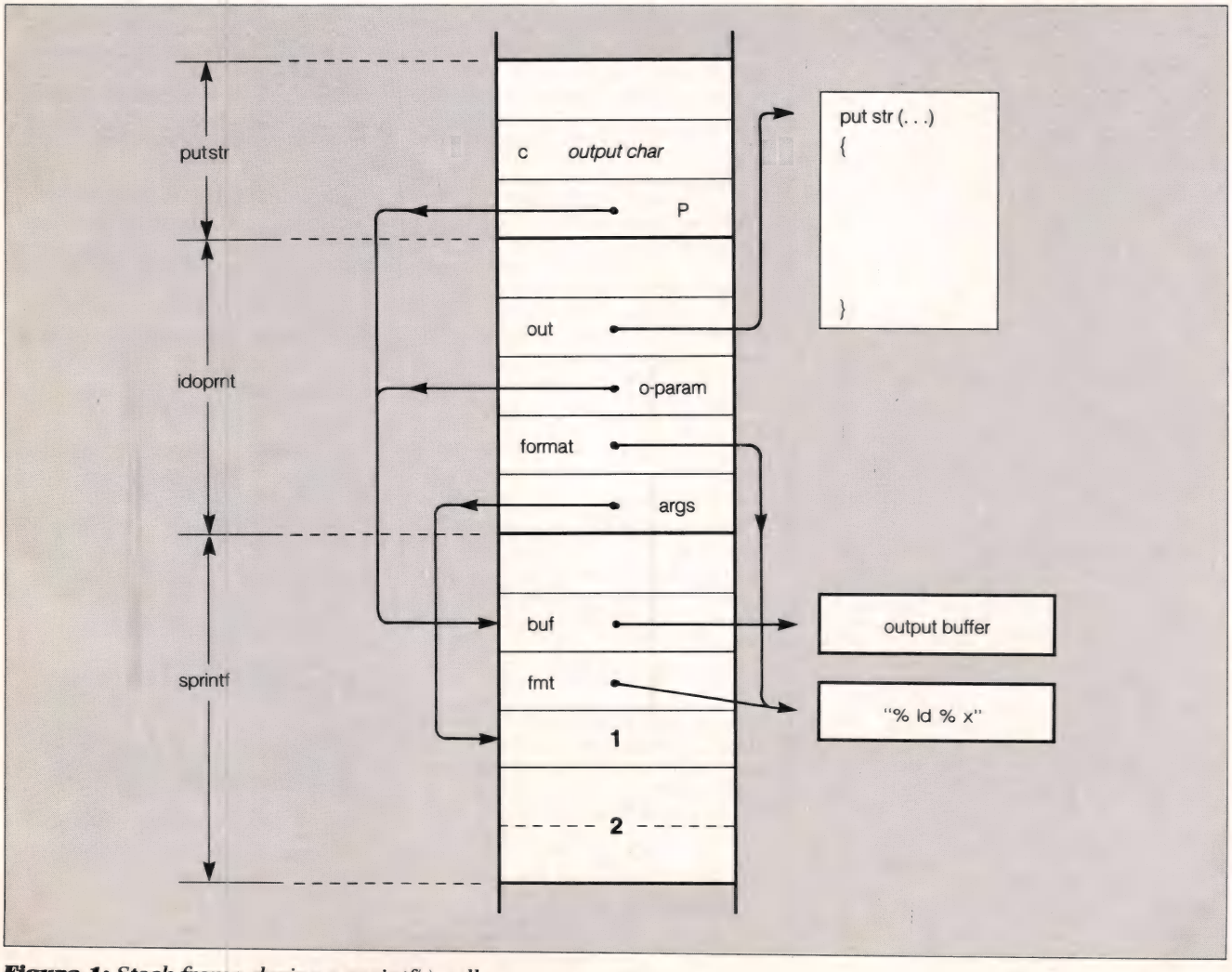


Figure 1: *Stack frame during a* sprintf( ) *call*

the conversion yourselves, the converted characters would be in *nbuf*, and *bp* would point at the end. You can subtract the two pointers to get the length.

The remainder of the loop just prints the converted string along with any necessary leading or trailing padding. You then loop back up to get the next conversion character. The advantage of this approach is that you need only enough local buffer space to take care of the largest possible numeric conversion, as compared to a worst-case buffer for the entire line.

The remainder of the file is just a test routine that makes sure everything works.

Ltos( ) is a *long-int*-to-string conversion routine. It's in Listing Two, page 71. There are only two things of interest. First, I'm building the string from back to front to make the conversion easier. The string is reversed in place on lines 54–61. Next is the somewhat weird-looking line 46:

```
*++bp =
"0123456789abcdef"[ n % base ];
```

The string *"0123456789abcdef"* evaluates to an rvalue of type *pointer to char* and the square-bracket notation can be applied to any pointer. For example:

```
"0123456789abcdef" [ 5 ];
```

evaluates to the character *5* (0x35).

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

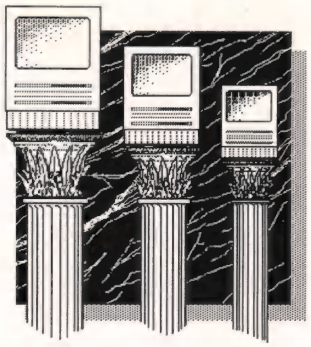
DDJ

(Listings begin on page 68.)

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

TO THE MACS

A Few Things That Work, Something That Doesn't, and a Little HyperTalking



by Stan Krute

In case you didn't catch my first column back in January, let me say it again: reviewing makes me uncomfortable. Creators put a lot of sweat into getting a product to market. Print is powerful stuff, and a bad review can do much harm. On the other hand, I like to share good news. So, in this column I'll talk about things I've used for a while and fundamentally like. Just call me Pollyana Krute....

On objectivity: I hate to crush any world views, but it's a myth. Product reviews are inevitably subjective. The best I can do is let you in on some of the experiential baggage I filter through. Also, I know and/or have worked with some of the people whose goodies I'll be reviewing. I'll always mention any such connections. Just know it's done to help you weigh my opinions, not as name dropping.

QUED/M 2.04

DDJ ran a nice article by Levi Thomas and Nick Turner on programmers and their text editors last year (February 1987). The article suggested that programmers might be a bit like baby ducks, bonding strongly with the first warm editor they meet. So, to help you assess the forthcoming opinions, my first computer-assisted writing was on IBM card punches and Teletype paper-printing terminals. Though primitive, both tools sported evocative sound effects.

Editing sounds have devolved, but text hacking's come a long way. I've done a lot of writing in succeeding decades, and currently work with several text-editing programs. My current favorite is QUED/M 2.04. It gives me power, speed, ease of use, and a smooth-sloped learning curve. Though I briefly mentioned the product back in column 1, I'd like to give a few more details.

Got a bad case of featuritis? QUED/M's got almost all the ones I've ever seen and/or wanted and a few more. A nonexhaustive list is:

- a powerful macro language, complete with a real-time recording mode
- high levels of user-configurability
- undos up to 32,767 (love that number) actions deep, depending on your RAM resources
- powerful regular expression and metacharacter facilities
- text folding
- line sorting
- ten clipboards
- automatic saves to two directories
- automatic saves after a configurable number of keystrokes
- backups of previous versions of a saved file
- adjustable scrolling parameters
- a full set of capitalization/casing commands
- multiline horizontal motion by tabs or spaces
- gremlin zapping
- windows with mucho status information
- window tiling and stacking commands
- user-configurable transfer menu
- parentheses balancing, with user-designatable definitions of "parentheses"
- the ability to open many files simultaneously
- multifile search/replace operations
- windows with horizontal and vertical split-paning
- text markers
- intelligent extensions to Apple's standard file-opening dialog

And on and on.... I sense QUED/M's designers have used a lot of text editors and Mac applications and have kept lists of features they've liked, detested, and wished for. And, unlike some applications that sport an abundance of commands and options, QUED/M's design and implementation seem clean, well organized, and intuitive.

There are three things I'd like in a future QUED/M:

- the ability to extend the command set via user-written standard Mac CODE resources
- the ability to work with files that don't fit whole hog into RAM
- a thicker manual, with slower/deeper explication and many more examples, particularly in the discussions of regular expressions and macros

Victor Romano programmed QUED/M 2.04. He and Jerzy Lewak designed it. Jerzy also wrote the manual. Two people, one fine text-editing product. Kudos, gentlemen, and thanks. I wish I had QUED/M in all my other computing worlds.

TMON 2.81

Here's more detail on another product I mentioned briefly back in column 1. TMON 2.81 is the Mac debugger I use all the time. It's small and rugged and works on a wide range of code resource types in a variety of complex environments.

TMON 2.81 provides a full range of standard debugging features. It runs in a simplified windowing environment that won't go down if your code munches critical parts of the Mac's regular windowing operations. In a variety of ways, you can:

- examine and change memory at

PAINLESS WINDOWS.

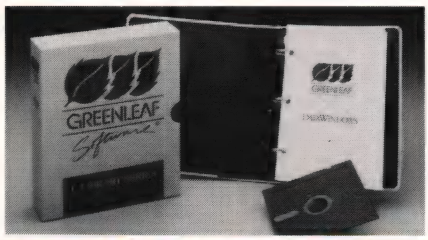
Windows. Data Entry. Menus. Finally, a C programmers' tool that makes them as easy to use as *printf( )*. With Greenleaf DataWindows™, you move in quantum leaps!

Snazzy Window Treatments

DataWindows represents an important breakthrough in C programming tools. It sets you free so you can create exciting programs quickly and easily, saving both time and money! Developed to work with the IBM PC, XT, AT, compatibles, and MSDOS or PCDOS, DataWindows is a carefully tooled system of C functions which will jazz up your programs with unprecedented efficiency.

Greenleaf DataWindows is integrated windows, transaction data entry, pop-up, pull-down, and Lotus style menu systems with:

- **Screen Management.** You don't have to remember what's on the display or the sequence in which you put it there. DataWindows does the grunt work. There are no restrictions.
- **Transaction Data Entry.** Data entry windows can have any number of fields with sophisticated options for reading many data types. Calls are made to help, validation, and other functions. Full featured text editing, protected and mandatory fields, dBASE type picture strings, context sensitive help, validation of fields and transactions, redefinable keys, password entry, attribute control, keyboard idle and much more.
- **Device Independence.** It detects the type of display adapter your computer is using and adjusts to it automatically for CGA, EGA, or monochrome. Logical video attributes are easy to use for color or monochrome.
- **Compatibility.** Runs with Microsoft Windows and IBM TopView.
- **The Greenleaf Tradition of Quality.** Reliable products. Professional documentation that gets you up and running quickly and keeps you there. Reference card. Newsletter and Bulletin board.



Stop Window Shopping

Order Today. Or call toll free for a free demo of the windows library that makes all the others obsolete.

Order any of these high performance tools by calling your dealer or 1-800-523-9830 today. Specify compiler when ordering. Add $8 for UPS second day air, or $5 for ground. Texas residents add sales tax. MasterCard, VISA, P.O., check, COD. In stock, shipped next day.

Greenleaf DataWindows	$225
DataWindows Source Module	$225
The Greenleaf Comm Library v2.0	$185
The Greenleaf Functions v3.0	$185
Digiboard Comm/4-II	$325
Digiboard Comm/8-II	$535

GREENLEAF Software©

16479 Dallas Parkway, Suite 570
Dallas, TX 75243

Call Toll Free
1-800-523-9830

In Texas and Alaska, call
214-248-2561

Window Dressings

- **Simple or Complex Windows.** Up to 254 powerful overlaid windows simultaneously, all with just one kind of window to remember! Yet any window can be from one character to 32K!
- **Easy Window Operations.** DataWindows lets you move, zoom, frame, title, change colors, titles, frames, size, location, and make windows visible or invisible at will! Functions set cursor, attributes, and write data to any window or "current window". Word wrap, auto scroll, keyboard functions.
- **Write to Any Window Any Time.** Windows may be visible, overlaid, or invisible, and you can write to them anyway. What you write will be seen when the windows become visible.
- **DataWindows is fast!** It writes directly to video memory (in some modes).
- **Easy to save!** Any window, complete with attributes, can be saved on disk quickly and efficiently.
- **Source code available. No royalties.**

Also from Greenleaf:

The Greenleaf Functions v3.0
The most complete, mature C language function library for the IBM PC,XT,AT and close compatibles. Includes over 225 functions — DOS, disk, video, color text and graphics, string, time/date, keyboard, disk status and Ctrl-Break functions plus many more.

The Greenleaf Comm Library
Our 2.0 version is the hottest communications facility of its kind. Over 120 functions — ring buffered, interrupt driven asynchronous communications for up to 16 ports simultaneously with XMODEM, XON/XOFF, many many sophisticated features.

We support all popular C compilers for MSDOS/PCDOS: Microsoft, Lattice, Computer Innovations, Aztec, DeSmet, and others.

TO THE MACS
(continued from page 106)

several levels of abstraction
- stop and start code execution
- fiddle with heaps
- convert and resolve quantities and expressions relevant to Mac programming work
- customize the program by extending its command set
- recover from crashes

There is one glaring omission in this version of TMON: though it won't blow when used with a 68020 or 68881, it won't disassemble non-68000 instructions or display non-68000 registers. So you can't really debug anything involving 68020s, 68881s, 68851s, et al. Shame, shame.

TMON was originally developed as an in-house tool by ICOM Simulations, whose other products include the graphic adventures Deja Vu and Uninvited. Waldemar Horwat (not a pseudonym, he actually exists) wrote the bulk of the program, albeit at a frighteningly young age. Darin Adler, currently working for Apple's

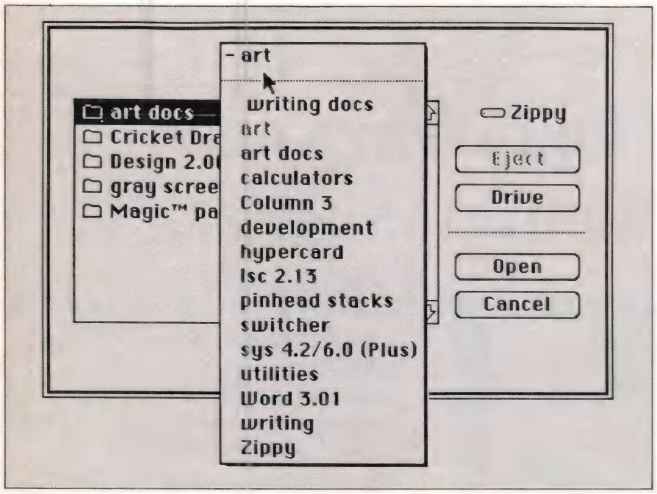


Figure 1: *A click on the directory name box in a standard file dialog brings up HFS Navigator's handy list of favorite directories.*

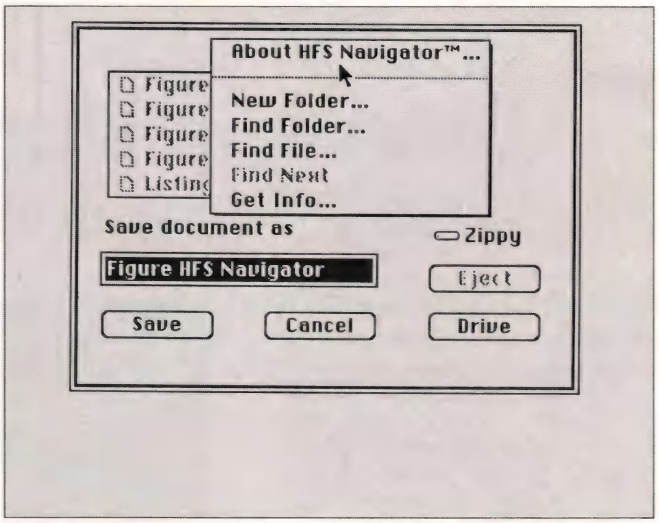


Figure 2: *Pressing the Command key causes Navigator to bring up its action menu.*

expanding Mac Tech Support team, did the set of command extensions (known as the User Area) that accompanies this version of the debugger. The new manual is quite good; Paul Snively wrote the users' guide section, and Waldemar wrote the technical reference part.

TMON may face some rough marketing pressures over the next few months as Think, Borland, and Apple bring out new source-level debuggers. TMON works primarily at the assembly level, though it has rudimentary source code label capabilities. But those other tools aren't out yet. And, even when they are, TMON should still be useful as a robust common denominator tool, especially when it's upgraded to full 680x0 support.

HFS Navigator

HFS Navigator is a small utility that Michael Kahl, chief programming force behind Lightspeed C, showed me on my September visit to Think central. It hooks into the Mac's standard file-opening and saving dialogs and lets you hop quickly to favorite directories on file opens and saves. It also finds files and directories and can create new directories on the fly.

That's not a bad set of features, but I already had desk accessories that pulled off the last three functions, and I've become quick and comfortable mousing up, down, and sideways through my carefully organized HFS directories. Sure, Navigator looked like a nice hack, but I didn't think I needed it.

I was (eventually) wrong. I tried Navigator for a week, gave it up, came back in a day, and now I'm hooked. You know how it is: a little taste of Mac ergonomics only makes you hunger for more, and HFS Navigator feeds the pangs.

A simple installation program lets you add Navigator to your HFS disk. Thereafter, whenever you mouse-press on the directory button in a standard HFS file dialog, HFS Navigator pops up a special menu of your favorite directories (see Figure 1, page 108). You just select the directory you want to go to, and you're there, without having to mouse up and down the branches of the HFS directory tree. The current direc-

PANEL® *Plus*

Advanced Screen Manager

Building an interactive application in C? **PANEL** *Plus* provides the features you need for professional program development:

PRODUCTIVITY

The **PANEL** *Plus* interactive screen design tools are the fastest way to get your application screens set out and tested. Just type prompts on the screen, mark out entry fields, define display and entry attributes, help boxes, borders, pop-up areas. Fields can be edited, moved and resized, and validation details entered – with the screen displayed so that you can see the effect of your changes.

Then **PANEL** *Plus* saves your work to disk, from where you can either load the design directly into your application program, or for better control, automatically generate C data structures, field areas, and header files which are compiled and linked into your program.

QUALITY

PANEL *Plus* screens can include all the features demanded by today's applications. Several different menu types are provided, including highlighted bars with help lines. Easy-to-use library functions support pop-up fields, horizontal and vertical scrolling in a field, and validation exits for supplied or custom data checking functions. Text functions can also be carried out in graphics mode using a supported graphics function library.

EASE OF USE

Although the library contains over 150 functions, it is logically organised so that most programs will only need to use a small subset. Documentation is provided with examples of all the main function calls. **PANEL** *Plus* includes full library source, with variant files for all supported systems, and no royalties are payable for the use of **PANEL** *Plus* libraries when linked into user applications.

PORTABILITY

PANEL *Plus* is designed to allow your programs to be ported to just about any environment where you can find a C compiler. Every version of **PANEL** *Plus* includes source modules for interfacing to: DOS, OS/2 protected mode, Amiga Intuition, Unix (with and without *termcap* or *termio*), Xenix, DOS-J (including 16-bit character editing), and VAX/VMS. Graphics libraries supported include MetaWindow, HALO, Essential Graphics, Microsoft C V5, and Turbo C V1.5. The Microsoft mouse can be used in PC versions.

Now available !

Roundhill announces screen tools for use with the new C compilers from Borland and Microsoft. Special introductory prices:

PANEL/TC – $129.00
For use with Borland's Turbo C

PANEL/QC – $129.00
For use with Microsoft's Quick C

(after May 1st 1988, $149.00)

Each package is configured for use on an IBM PC or compatible system, and screens can be designed and built into your programs while running in the special development environment provided with the compiler.

All the **PANEL** *Plus* library functions are supported, and source code for every validation function is provided so that you can customise the entry checking to suit your application. When you need to move your programs to other environments, or need the full library source for other reasons, upgrades to **PANEL** *Plus* are available.

PANEL/QC can be run in any of the graphics modes supported by Quick C, and also interfaces to Microsoft C V5. PANEL/TC fully supports the Turbo C 1.5 graphics library.

PANEL *Plus* for MS-DOS or for OS/2, with full library source, is priced at $495.00. Versions are available for the Aztec, Borland, CI C86PLUS, IBM, Lattice, Mark Williams, MetaWare, and Microsoft compilers. Please call for prices of **PANEL** *Plus* for Xenix and Unix systems, and for VAX/VMS. Existing registered users of PANEL will receive a credit against the **PANEL** *Plus* license fee.

Roundhill Computer Systems Limited **(201) 569 2265**
PO Box 8107, Englewood NJ 07631

Roundhill Computer Systems Limited **(0672) 54675**
PO Box 14 Marlborough SN8 1LR England

Telex (UK): 444453 AWARE G **BIX:** *join roundhill*
Fax (UK): (0672) 54436

Roundhill Computer Systems

tory, shown at the top of Navigator's pop-up menu, can either be added to or deleted from the list of favorites.

If you hold down the Option key when you mouse-press the standard file dialog's directory button, up comes the standard pop-up menu, showing the path to the current disk's root directory. And, if you hold down the Command key during the mouse-press, a master menu pops up that lets you find files and directories and create new folders (see Figure 2, page 108). Folder making at such an opportune moment is very useful.

I do have two things I'd like the makers to tweak, in the name of even greater ergomania. First, as currently configured, the menu of favorite directories is limited to 16 entries. I keep bumping up against that ceiling. My solution would be to raise the limit and, to keep user access fast, arrange the entries in two dimensions, as a table rather than as a list. See Figure 3, page 113, for my vision.

Second, I don't like having to hold down the Option or Command keys when I want the standard path-to-root or master pop-up menus. A possible solution would be to give a mouse-press on one side of the directory button to get the favorites, a press on the other side to get the path-to-root, and a press within a few pixels of the top of the directory button to bring up the master menu.

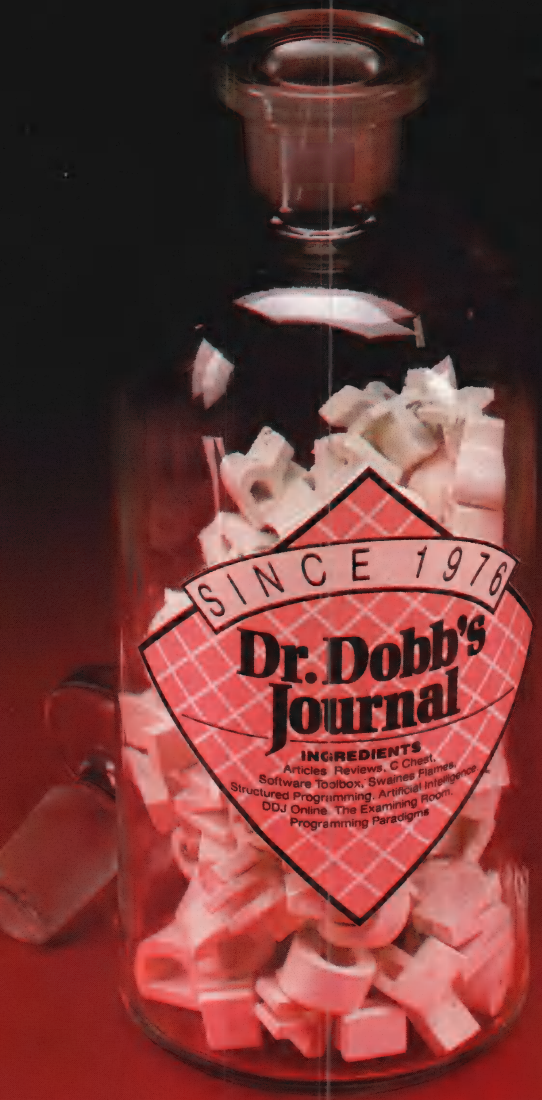
HFS Navigator has a competitor, FindSwell, but I haven't got my hands on it yet. When I do I'll report back. Meanwhile, as I said earlier, Navigator's got me hooked.

WYSIWYG Ain't

I do like to maintain a positive attitude. So view the next few paragraphs as an opportunity for fabulous personal financial growth. See a need and fill it, as the kitty litter and blue toilet water inventors say.

Here's the situation: I recently spent three weeks doing production desktop publishing with a friend who's a commercial printer. I was also involved in producing the first issue of another friend's desktop-

THE CURE FOR COMMON CODE



Are you getting the recommended monthly allowance of C, Assembly, Forth, Pascal, BASIC or Modula-2? Subscribe to *Dr. Dobb's Journal of Software Tools* and you won't catch any nasty bugs again!

Each month the Doctor brings you aid for ailing algorithms and the cure for common code. For the latest developments in software design and pages of code that will make you a more productive programmer, take the Dr. Dobb's prescription.

For more than a decade, the programming elite have known *Dr. Dobb's Journal* to be the foremost source of software tools. Subscribe now and get your monthly dose from the Doctor.

PASCAL
PROLOG FOR
C BASIC
ASSEMBLY
MODULA-2

$5 SAVINGS

SUBSCRIBE AND SAVE!
Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS
and save over $5—a 15% savings off the cover price!

☐ 1 year $29.97 ☐ 2 years $56.97

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

ONLY $29.97! YOU SAVE OVER $5.00

*Savings based on the full one-year newsstand rate of $35.40. All foreign countries please add $11 per year for surface mail; Canada & Mexico add $28 per year for airmail; other countries add $32 per year for airmail. All foreign subscriptions must be paid in U.S. funds drawn on a U.S. bank. Please allow 6-8 weeks for delivery.

A Publication of M & T Publishing, Inc.

363S

COMMENTS & SUGGESTIONS

April 1988, #138

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail. —Ed

Which articles or departments did you enjoy the most this month? Why?

_____

_____

_____

Comments or suggestions _____

_____

_____

_____

Name: _____

Address: _____

(continued from page 111)

published magazine. And I just finished putting out my own little software company's latest quarterly catalog. In all three instances Macs were used, and the overall process and results were quite satisfying. But we repeatedly came up against an irritating and needless bugaboo, one that creates the incredible opportunity noted earlier.

WYSIWYG is a myth. What you see on the Macintosh screen *is not* what you get on the laser-printed page. Oh, it's close. Frustratingly close. But not close enough for real commercial production work. Hairlines vary in size. Text and graphic elements lose their relationships. Whole sentences can switch pages. On-screen measurements indicate you're controlling placement to four or five decimal places, and the finished result can vary by big pieces of inches. And this is true, to varying but never completely insignificant degrees, with *every* piece of desktop software we used. So you proof and kludge and proof and kludge until the result's acceptable.

There's no good reason for such behavior. These are computers, kids, and they're very good at arithmetic. The Mac has SANE, with ungodly levels of precision. If software SANE's too slow, you can write directly to the hardware. If Apple's LaserWriter driver doesn't work properly, just send PostScript out directly. If PostScript's the problem, learn how to work around it. But, please, somebody, do something. This is intolerable.

Deliver desktop publishing accuracy along with ease of use and automated power, and you'll get very rich. 'Cos there are a lot of people out there doing this stuff, and they all know the truth: the WYSIWYG emperor's walking around stark jaybird nekkid!

Code Corner

I spent some time this last month wandering around HyperCard and HyperTalk. Serendipitous travel's my most fruitful learning mode. This month's code corner features a small toolkit I built to aid that exploration. Building and running it revealed some of HyperCard 1.0.1's contradictory qualities.

First, the script editing facilities can be politely described as primitive, but the graphics editing facilities are elegant.

Second, execution speed can be slow. Depending on the hardware, this month's project takes 0.5 to 4 minutes to copy less than 20 objects from one stack to another. Third, execution speed can be fast. My buddy Bruce The Q. Hammond built a 40,000 card HyperCard database (he likes to break and fix things). Searches over this multimegabyte file were in the sub-30-second range.

Fourth, HyperTalk syntax can be intuitive. Right from the start I could write large pieces of code without continual manual browsing. Fifth, HyperTalk syntax can be inconsistent—for example, the usage of the word the. Sometimes it's optional, sometimes it's mandatory, sometimes it's forbidden. Aargh! Who can keep track?

Sixth, HyperCard can be used to build large, complex coordinating tools. Take a look at some of the commercial applications starting to appear.

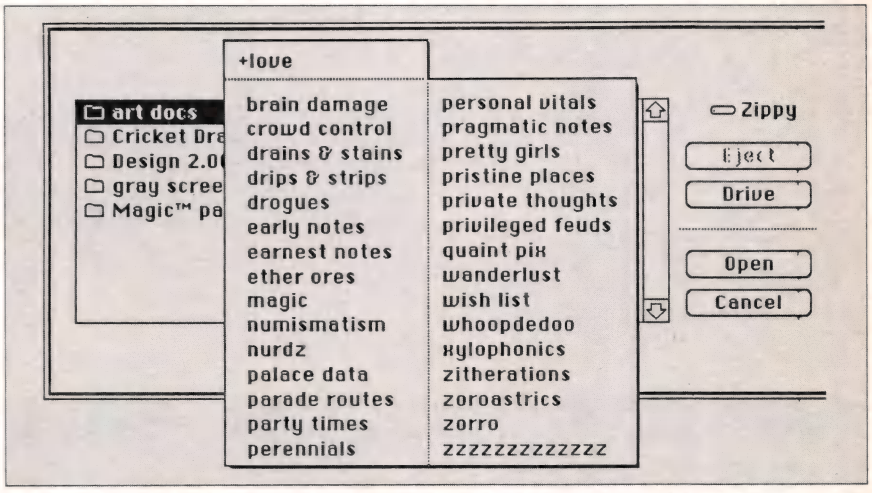


Figure 3: *Author's vision of an improved Navigator, stretching the user interface along an orthogonal path*

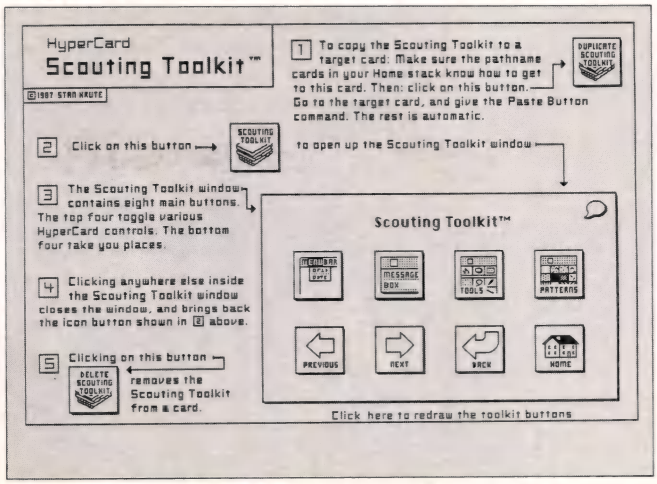


Figure 4: *The Scouting Toolkit's card of residence*

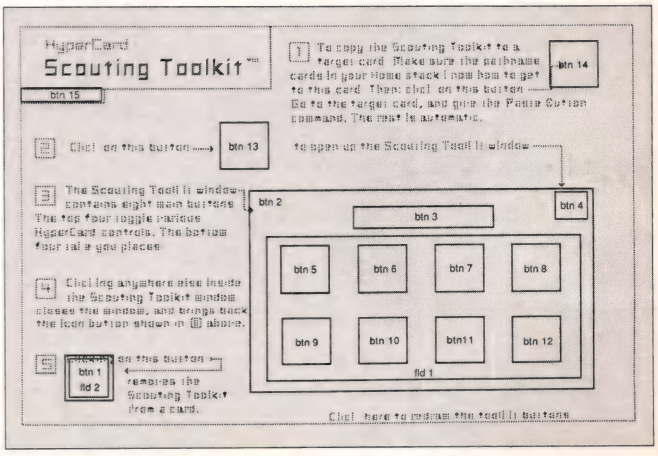


Figure 5: *Where the 15 buttons and 2 (hidden) fields live on the toolkit's residence card*

OS/2 UNIX

New VCScreen 2.0

■ WINDOWS ■ DATA ENTRY ■ MENUS ■ HELP MANAGEMENT ■ TEXT EDITING ■

Vitamin C

PROFESSIONAL C LANGUAGE FUNCTION LIBRARY

- ❑ Multiple bullet proof overlapping windows
- ❑ Easy single field or full screen data entry
- ❑ Unlimited data validation
- ❑ Context sensitive help manager
- ❑ Menus like Lotus & Mac
- ❑ Programmable keyboard handler
- ❑ Text editor routines
- ❑ Printer output routines

- ❑ 30 day money back guarantee
- ❑ No royalties or runtime fees on applications
- ❑ Complete library source code included FREE
- ❑ FREE technical support
- ❑ FREE BBS at (214)418-0059
- ❑ Supports Microsoft 5, Quick C, Turbo C, Lattice and others
- ❑ Optional screen painter/generator

Better Applications In Less Time

Fast, flexible, versatile, reliable. Just some of the reasons why serious programmers use Vitamin C in their most important projects. They know using Vitamin C means lightning fast displays, a responsive user interface, professionally crafted C code, and a commitment to technical support.

High level functions provide maximum speed and productivity. Extended versions of these same routines add flexible control over specific details when necessary.

Generate Code Inter

NEW VERSION!

Speed development even more with VCScreen, our interactive screen painter / code generator. Define windows, boxes, borders, headings, input and output fields. Copy, delete, change, move, even layer objects. Then let VCScreen generate C source code ready to compile and link with the Vitamin C function library.

New features allow creation of multiple windows, menu systems, global variable maintainance, user defined code generation options, and more user configuration options!

Versatile Design Keeps You In Control

Options and possibilities rather than limitations and frustrations mean you're always in control. Our versatile open ended design is full of hooks so you can intercept and plug-in your own control functions to easily customize or add features to most routines.

Easily create windows that pop-up, overlap, zoom, move, scroll, hide, show and resize. You'll choose options for titles, borders, colors, scroll bars, virtual size, and more. You can even access any window any time, even if it's hidden or invisible. That's flexibility.

Sophisticated data entry forms become easy with features like unlimited validation, protected, invisible, and scrolling fields, full color control, single and multiple field input, selection sets, even right-to-left numeric input! And, with the context sensitive help system it's easy to provide field specific or other help messages.

Vitamin C's menus are the perfect framework for any application and feature advanced options such as check marks, unavailable items, blank items and separators.

The keyboard handler routines can redefine key assignments, translate keystrokes, even call a function.

Utility routines for time/date management, background processing, and sending windows to a printer.

Thorough documentation with tutorial and reference sections. Reference databases compatible with the Norton Guides Instant Access Program are also available.

Users And Reviewers Agree

" Picking the best value package is hard... If you' re a source code fanatic like me, Vitamin C is preferable. If you need source code, make sure your wallet is wide open or get Vitamin C. "
Computer Language, June '87

"Only Vitamin C supports keyboard handlers and keyboard reassignment. Vitamin C provides the most options for menus."
BYTE, October '87

"I trust our review of [Vitamin C] in Computer Language magazine was fair . . . it has become the screen manager package of choice at my firm."
Michale Wilson, Wilsoft, Inc.

OS/2, UNIX and Xenix versions now available. Call for prices and details.

Vitamin C.....$225.00
Includes source. Specify compiler when ordering.

VCScreen.....$149.00
Requires Vitamin C library above.

Reference Database... $50.00
Requires the Norton Guides program sold separately.

Requires IBM PC, XT, AT, PS/2 or compatible. Include UPS shipping: $3 for ground, $6 for 2nd day air, $20 for overnight, $30 if outside U.S. All funds must be in U.S. dollars drawn on a U.S. bank.

ORDER NOW!
(214)
416-6447

creative PROGRAMMING

Box 112097 Carrollton, Tx 75011

CIRCLE NO. 187 ON READER SERVICE CARD

And finally, HyperTalk can bog down while running through large amounts of complex code. Take a look at some of the commercial applications starting to appear.

Notice the seesaw here. It's good, it's bad. It's fun, it's irritating. It breaks Macintosh conventions, it extends the Macintosh metaphor. Hmm. . .final judgments will obviously have to wait. Let's just hope Bill and his team keep at it 'til they get it right. Meanwhile, on to the project.

Getting a Toehold

As I learned long ago from the Kernighan/Ritchie/Plauger crowd, the first thing you want to do in a new environment is build some simple tools, then lever yourself up into power and sophistication. HyperCard's no exception to the rule.

Whenever I started work on a new stack or card, I found myself spending quite a bit of time bringing in the basic buttons that let me hook into HyperCard's facilities. I'd find myself unable to go Home or without a menu bar. And I hate memorizing key combinations. So, I built a useful set of buttons that can be installed with one click and one paste—automatically. It's the Scouting Toolkit.

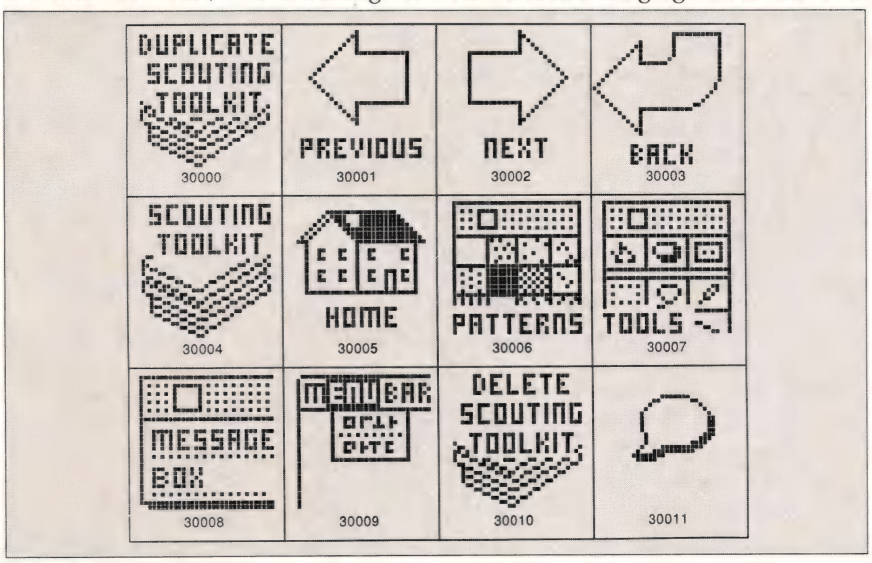


Figure 6: *Design details of the 12 icons used in the Scouting toolkit*

Toolkit Descriptions and Behaviors

Figure 4, page 113, shows the self-documenting card the toolkit lives on. Figure 5, page 113, identifies the 15 buttons and two (hidden) text fields that make up the domicile card's object world. All but one field and one button go with the toolkit when it travels to new stacks and cards. Figure 6, left, details the 12

icons used in the buttons. Listing One, page 72, gives complete descriptions and scripts for the card, its stack, the buttons, and the fields.

Installing the toolkit on a new card is easy. First, make sure your Home card knows where the toolkit's stack lives. Then click on button 14. It copies itself to the clipboard. Now go to your target stack and card, and give a Paste command. As ceaselessly mentioned, the rest is automatic.

The toolkit proper contains eight major buttons, numbers 5 through 12. The top four let you toggle various environmental windows. The bottom four take you to useful places. The other toolkit buttons support the big eight.

The whole toolkit folds up and disappears if you click in its interior outside the buttons. It reinflates when you click on its iconic remainder, button 13.

Like all good organisms, the toolkit knows when to fold its tent and scoot off into the night. Button number 1 removes all traces of the toolkit from a card, including itself. Snake tail swallowing is one useful byproduct of HyperTalk's ability to self-reference.

Trees Saved As Description's Delayed

I keep telling Tyler I'll cut down the size of these columns and their listings. And I'm trying. But once again I've exceeded my spatial budget. So you'll have to wait until next month for the remaining discussion of the Scouting Toolkit's objects and their operation.

Of course, Marvel and DC always come up with dramatic catchphrase hooks for their continued graphic stories. But all I can think of is this small section's subhead. Who ever said comic books ain't superior literature?

Wrap-Up

I've worked without reader feedback on these first few columns—frightening but true. So, if you get a chance and have the interest, drop me a note detailing what you want to see more and less of.

Next time out, besides the project wrap-up: reader mail, Mac Expo, parentheses surrender, the Cambridge ambience, intelligent pictures, Microsoft madness. And, of course, another code project. See you in 31.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 221. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 72.)

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

Vendors

HFS Navigator
Think Technologies
135 South Rd.
Bedford, MA 01730
(800) 648-4465
(617) 275-4800
Reader Service No. 20

QUED/M 2.04
Paragon Concepts Inc.
4954 Sun Valley Rd.
Del Mar, CA 92014
(619) 481-1477
Reader Service No. 21

TMON 2.81
ICOM Simulations Inc.
646 S. Wheeling Rd.
Wheeling, IL 60090
(312) 520-4440
Reader Service No. 22

A DEAL YOU CAN'T REFUSE

FREE* Turbo C® (Borland) or **FREE* QuickC™** (Microsoft) or **Microsoft®C $100 REFUND** on your present MS C compiler with purchase of C Starter or C Business Library

** If you ALREADY own one, get a **FREE REFUND**... See special offer.* (Limited time offer)

C *WHY* YOU CAN'T REFUSE

A C COMPILER *without a good add-on library is like a PC without a keyboard... it won't do what you want it to do.*

GAIN C POWER *Add capabilities your compiler library does NOT have. e.g.:*

- *New! Qwick Menuing—full 1-2-3 like menus & more*
- *Flexible powerful windowing + new Qwick windows*
- *Powerful cursor, video and attribute control*
- *Time and date arithmetic*
- *Sample code and working examples*
- *New! Qwick Data Entry with dialog boxes*
- *Formatted, fully validated data entry*
- *Display default field values*
- *Calculator style entry option*
- *500 functions you need*

SAVE TIME, TIME, TIME: *man-years on development, calendar months on schedule!*

SAVE MONEY: *Lowest Cost, Highest Quality Library/Windows Available!*

SMALLER PROGRAM SIZE: *your application program can be up to 50% smaller!*

EASY *for beginners!* **POWERFUL** *for professionals!*

INSTANT INSTALLATION UTILITY *included!*

SUPERB DOCUMENTATION: *time saving, helpful, clear, complete, instructive.*

BUSINESS USERS: FREE *3 machine site license (C Library & Power Windows).*

FULL SOURCE CODE! NO ROYALTIES *on products you develop.*

FREE UTILITY: *To convert Turbo Pascal code to C code.*

BORN AGAIN DISCOUNT
*If you're not happy with your present tools for C, or would like to upgrade to Entelekon, ask about the **Big** Born Again Discount.*

SAVE MONEY! SAVE TIME! DON'T WAIT! ORDER NOW!

SATISFACTION GUARANTEED **(Direct from Entelekon only)** **CALL (713) 468-4412**

POWER WINDOWS™
MOST POWERFUL YET
POP-UP/PULL DOWN/OVERLAP
Menus/Overlays
Help Screens
Messages/Alarms
ZAP ON/OFF SCREEN
FILE-WINDOW MANAGEMENT
Horizontal & Vertical Scrolling
Word Wrap & Line Insertion
Cursor/Attributes/Borders
Many types of menus. Highlighting. Move data between files, keyboard, program and windows. Status lines. Change size/location/overlapping. Move/add/delete/cascade windows.
6 diskettes $159.95

C FUNCTION LIBRARY
BEST YOU CAN GET
HUNDREDS OF FUNCTIONS
FULLY TESTED
BETTER FUNCTIONS
Most complete screen handling plus graphics; cursor/keyboard/data entry, 72 string functions with word wrap; status and control; utility/DOS BIOS/time/data functions; printer control & more. Special Functions.
14 diskettes $159.95

C BUSINESS LIBRARY
INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C, B-TREE LIBRARY, ISAM
ALL for $299.95
(A $500.00 VALUE)

B-TREE LIBRARY & ISAM DRIVER
POWERFUL DATA MANAGER
FAST! EASTY TO USE!
16.7 MILLION RECORDS/FILE
16.7 MILLION KEYS/FILE
Fixed/Variable length records.
Fast B-tree indices. Add/remove keys Find first/last/next/any key. Find keys by Boolean selection. Read/write/delete or add records to file.
Full source. No royalties $129.95
Multi-User option available.

C STARTER PACKAGE
INCLUDES C FUNCTION LIBRARY, POWER WINDOWS, SUPERFONTS FOR C (20 DISKETTES)
ALL for $199.95
(A $370.00 VALUE)

* SPECIAL OFFER
Free Turbo C or QuickC with purchase of C Starter Package, C Business Library, C Function Library or Power Windows. Even if you already own Turbo C or QuickC, we will refund up to the full purchase price of one of these packages with the purchase of C Starter Package or C Business Library.

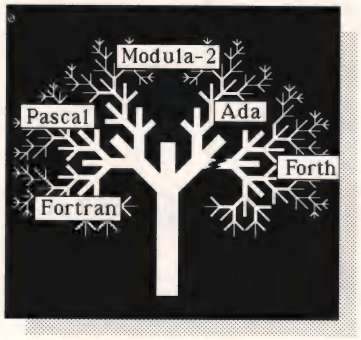
Entelekon™

SINCE 1982

12118 Kimberlev. Houston. TX 77024 713-468-4412 VISA-MASTERCARD-CHECK-COD

CIRCLE NO. 190 ON READER SERVICE CARD

Implementing Wirth's LineDrawing Module



In his seminal work *Programming in Modula-2*, the venerable Nicklaus Wirth, lord of structured programming, proposes a graphics module that he calls LineDrawing. It's defined on pages 114–115 of my copy, which is the Springer-Verlag 1982 edition. Although LineDrawing lacks the razzle-dazzle of more recent graphics packages, it's serviceable for many applications. Yet Modula-2 vendors seem to have bypassed it even though they leap at everything else Wirth even vaguely suggests. Consequently, this month's column makes a Valuable Contribution by implementing LineDrawing and showing some ways to use it.

For this project I tried out a new Modula-2 development system from Stony Brook Software up in Wilton, New Hampshire. I used Version 1.00, a level number certain to make any serious programmers break out in a sweat as cold as the New Hampshire winter. It's a command-line compiler, which isn't much fun after getting used to Turbo Pascal 4.0's glitzy integrated environment. Also, I didn't use the included program editor, M2EDIT, because I'm spoiled by BRIEF. (If you haven't tried BRIEF, it's like coming to the One True Religion.) Still and all, Stony Brook's Modula-2 is a terrific compiler. I found only one real bug using it for this and some other projects: remarkable for any new product. It's dazzlingly fast as well, compiling the 201-line LINEDWG.MOD in five seconds from Enter to system prompt on my 8-MHz AT clone with a 40-ms hard disk.

by Kent Porter

Lest the previous paragraph leave you with the impression that I'm just overflowing with praise for all and sundry, let me take a whack at The Master Himself. Professor Wirth might be the progenitor of structured programming languages, but he writes stylistically sloppy code. If this guy were a student in one of my programming classes, I'd drop his grade for the lack of comments and a tendency to put unrelated statements on the same line. Wirth gets away with it because he's Who He Is, but that's no excuse, particularly in a work that is to Modula-2 what K & R is to C. And while I'm at it, Springer-Verlag gets a hiss for consistently producing the worst-indexed books in the computer publishing industry.

So much for congratulations and contumely. Let's get on with it.

The Line-Drawing Module

The definition module for LineDrawing is LINEDWG.DEF in Listing One, page 88. With minor stylistic changes, it adheres faithfully to Wirth's definition. The only alteration of substance is the name itself; LineDrawing doesn't fit into the eight characters DOS allows, so I shortened it to LineDwg.

There's one other change as well. In the prototype, Wirth defines a procedure called *area*, which fills a rectangular region with a color. But elsewhere, in connection with the Queens program on pages 58–59 (which actually calls the procedure), he refers to an apparently identical process called *paint*. The latter makes more sense, so that's its name in LINEDWG.DEF.

I chose to implement LineDwg using EGA (the IBM Enhanced Graphics Adapter) mode *10h*, which furnishes 640×350-pixel color graphics. This is somewhat at odds with the Wirth model, which proposes four gray scales from white (0) to black (3), but it makes sense in that the EGA is widely available and it has decent resolution. Mapping Wirth's color indicators to EGA color numbers is easy to do via a *CASE* statement.

The problem with EGA 640×350 is that the pixels aren't square. To achieve orthogonal integrity on an

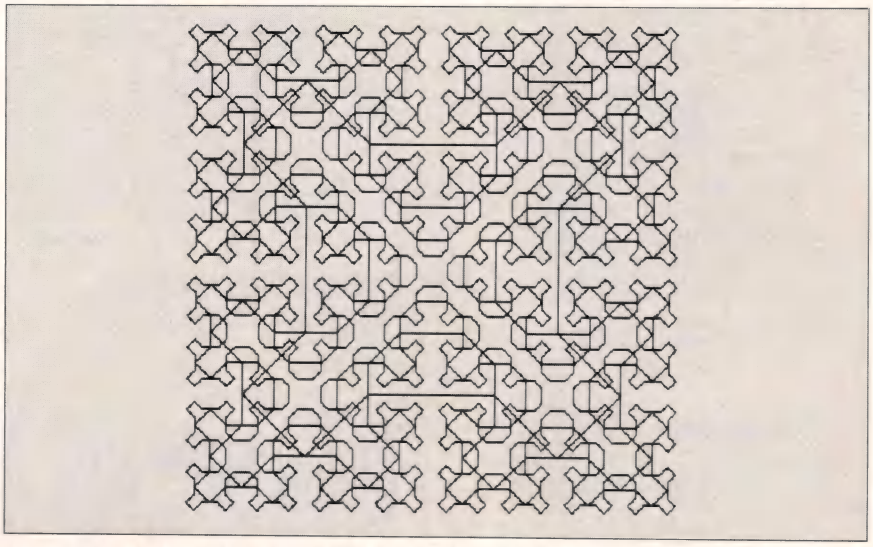
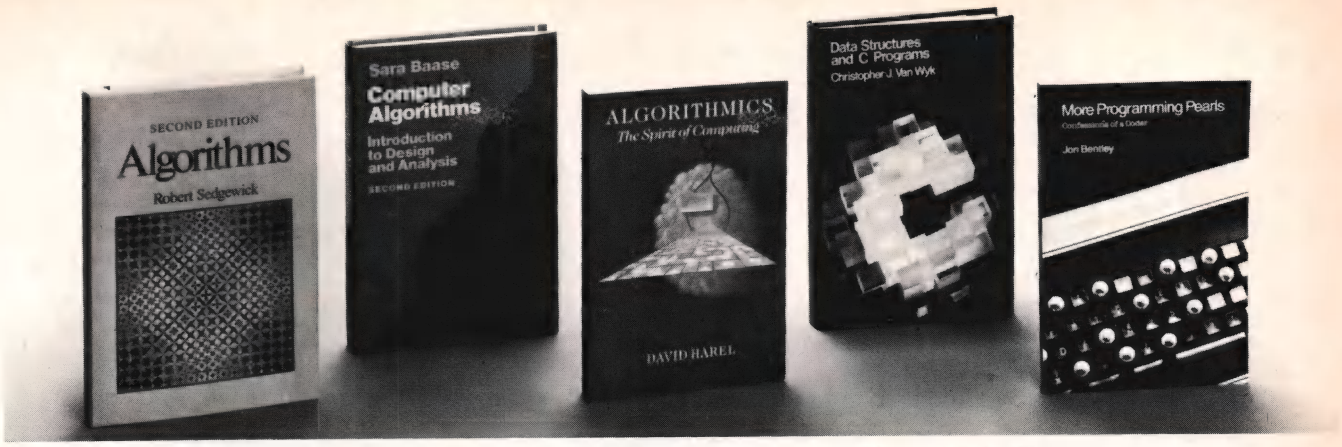


Figure 1: *Output from SIERPIN.MOD, drawn with LineDwg routines*



Five new ways to build better programs.

And one truly classic way.

What's new...

1. ☐ **Algorithms, Second Edition**
by Robert Sedgewick.
"Impressive..." said Byte magazine of the first edition. Right off the press, this second edition of **Algorithms** gives you the same broad-based coverage and emphasis on an intuitive approach to algorithm analysis as the extremely successful first edition.
06673 640 pp. Hardcover $37.75

2. ☐ **Computer Algorithms, Second Edition**
by Sara Baase.
Several new chapters on modern topics like parallel algorithms have been added to this edition of our popular book. And the solid mathematical analysis and clear, accessible style haven't changed one bit! Includes discussions of advanced topics like adversary arguments for lower bounds, and NP-completeness.
06035 400 pp. Hardcover $38.75

3. ☐ **Algorithmics: The Spirit of Computing**
by David Harel.
"A wonderful introduction to algorithms and complexity theory." (Alfred V. Aho, *director of Computing Science Research at AT&T Bell Laboratories, Murray Hill, NJ).* Here's a highly readable account of the most important concepts, methods, and results fundamental to the science of computing.
19240 423 pp. Paper $24.75

4. ☐ **Data Structures and C Programs**
by Christopher Van Wyk.
Here's your chance to own a unique book on data structures—with examples coded in the C language. Van Wyk gives you the "why" and "how" of program implementation, and includes only working, tested programs as his examples.
16116 600 pp. Hardcover $40.95

5. ☐ **More Programming Pearls**
by Jon Bentley.
Hot off the press! Here's more of the popular columnist's "pearls of wisdom," written to explore, teach, and entertain. Shows you how creativity and insight in computer programming can be applied practically in engineering.
11889 224 pp. Paper $16.25

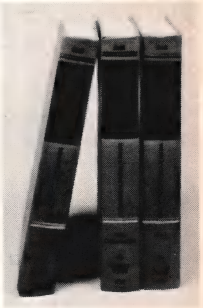
What's classic...

The Art of Computer Programming
by Donald E. Knuth.

☐ **Volume I: Fundamental Algorithms, Second Edition.**
03809 643 pp. Hardcover $45.25

☐ **Volume II: Seminumerical Algorithms, Second Edition.**
03822 200 pp. Hardcover $45.25

☐ **Volume III: Sorting and Searching**
03803 722 pp. Hardcover $45.25



TO ORDER: Simply check off the books that interest you, tear out this ad and send a check for the total, plus your local sales tax, to: Dept. DM, Addison-Wesley Publishing Co., 1 Jacob Way, Reading, MA 01867. Or call the order dept. at the number below and give the 5-digit code at the bottom of this ad.

Addison-Wesley Publishing Company
Reading, Massachusetts 01867 • (617) 944-3700

We publish the leaders.

16039

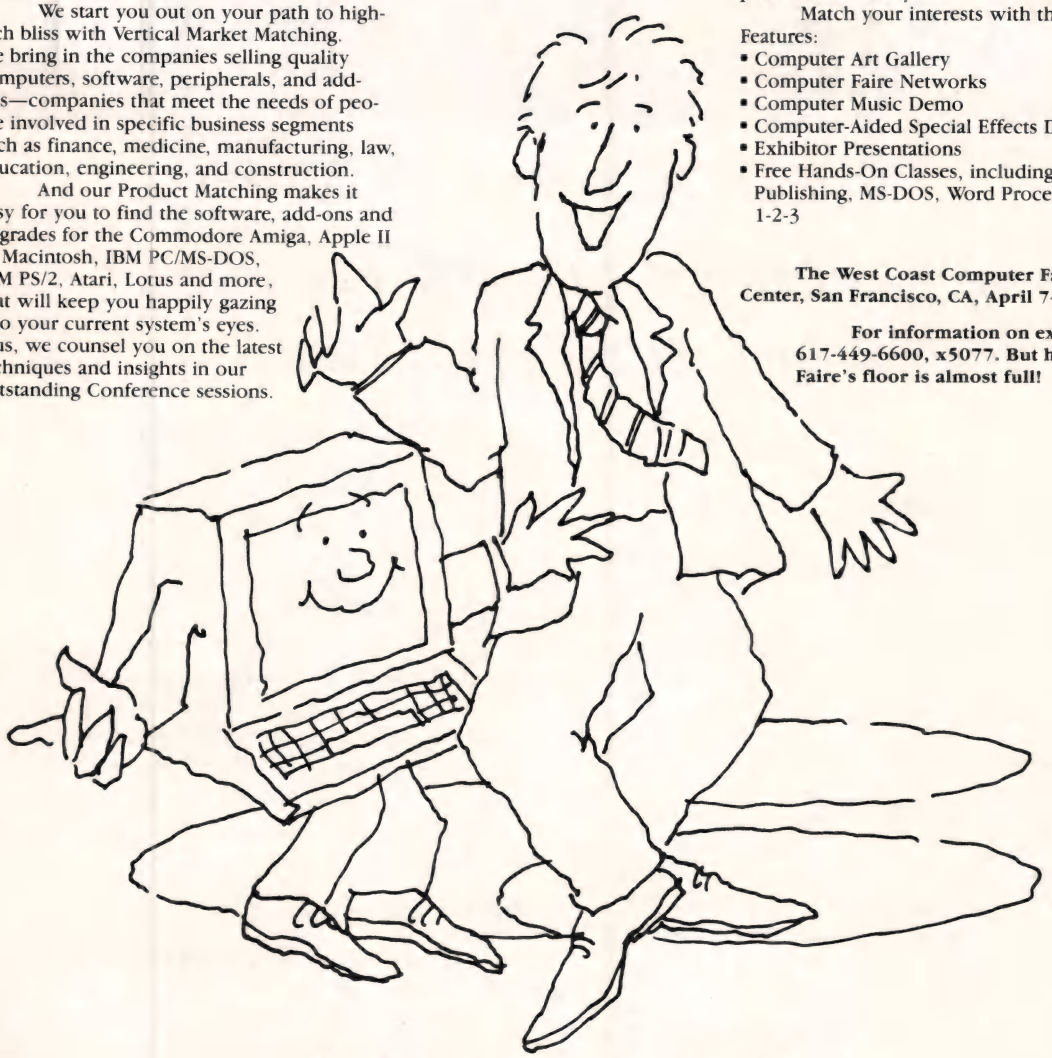
CIRCLE NO. 191 ON READER SERVICE CARD

The West Coast Computer Faire announces the first Computer Matchmaking Service.

You won't have to depend on fate at the 13th West Coast Computer Faire to find the products and services that are the perfect match for your needs.

We start you out on your path to high-tech bliss with Vertical Market Matching. We bring in the companies selling quality computers, software, peripherals, and add-ons—companies that meet the needs of people involved in specific business segments such as finance, medicine, manufacturing, law, education, engineering, and construction.

And our Product Matching makes it easy for you to find the software, add-ons and upgrades for the Commodore Amiga, Apple II or Macintosh, IBM PC/MS-DOS, IBM PS/2, Atari, Lotus and more, that will keep you happily gazing into your current system's eyes. Plus, we counsel you on the latest techniques and insights in our outstanding Conference sessions.



The West Coast Computer Faire has made and will make more matches than any another computer show. It's time we made the perfect match for you.

Match your interests with these Faire Features:

- Computer Art Gallery
- Computer Faire Networks
- Computer Music Demo
- Computer-Aided Special Effects Demo
- Exhibitor Presentations
- Free Hands-On Classes, including Desktop Publishing, MS-DOS, Word Processing, Lotus 1-2-3

The West Coast Computer Faire, Moscone Center, San Francisco, CA, April 7-10, 1988

For information on exhibiting, call 617-449-6600, x5077. But hurry — the Faire's floor is almost full!

Register early and save $5!

Fill out this coupon and mail with your check(s), for $15.00 for each registrant, postmarked by March 17th, 1988. Include the names and addresses of registrants for whom you are enclosing a check. (Photocopy coupon for additional registrants.)

Name _____ Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone (_____) _____

Four day conference and exhibits $15.00 in advance. $20.00 at the door. Make check payable to "The West Coast Computer Faire." Mail to: Attendee Registration Department, The West Coast Computer Faire, 300 First Avenue, Needham, MA 02194. Advanced registrations accepted only with full payment and each registrant's name and address. Tickets will be mailed to each individual registrant separately.

THE 13TH WEST COAST COMPUTER FAIRE

April 7–10, 1988, Moscone Center, San Francisco, CA

CIRCLE NO. 192 ON READER SERVICE CARD

(continued from page 118)

IBM PC display, the *y* dimension should be 75 percent of the *x*. The EGA resolution yields 53.8 percent of *x*, so everything is taller than it should be. Because Wirth's line-drawing procedure works in increments of 45 degrees, he obviously had square pixels in mind.

The solution is to use a virtual-coordinate space to represent the display. In virtual space, the screen can be 800 × 600 pixels, and following the Cartesian convention, the origin (coordinates [0,0]) can be at the lower-left corner with y ascending upward. Because the virtual range is larger than the device range, rounding off tends to jam pixels together, thus producing visual objects without gaps. The opposite effect—gapping—occurs when the virtual range is smaller than the physical, as in 400 × 300 pixels mapped to a 640 × 350-pixel display. In both cases you get jaggies, but jaggies are an inevitable feature of the computer graphics landscape. Fortunately, they're not glaringly apparent with the EGA.

Mapping virtual coordinates into physical space is relatively simple. The program works entirely within the virtual range, and only the output functions "know" that the coordinate system is a myth. The functional procedures *devX* and *devY* in Listing Two (the LineDwg implementation), page 88, perform virtual-to-physical translations based on factors computed during the module's initialization phase. These functions are called when writing pixels to the device. The only procedure exempt from virtual addressing is *writePixel*, which works entirely in device space.

Perhaps one reason why LineDrawing has not been implemented elsewhere is that Wirth's thoughts on the subject seem to be incomplete. For example, he formulates the *PaintMode* enumeration as a control for the *paint* and *copy* routines, yet none of the calls ever touches it. Consequently, it's set here to *replace* and never changed, and the routines don't refer to it. Likewise the matter of color inheritance is unresolved. The *paint* and *dot* proce-

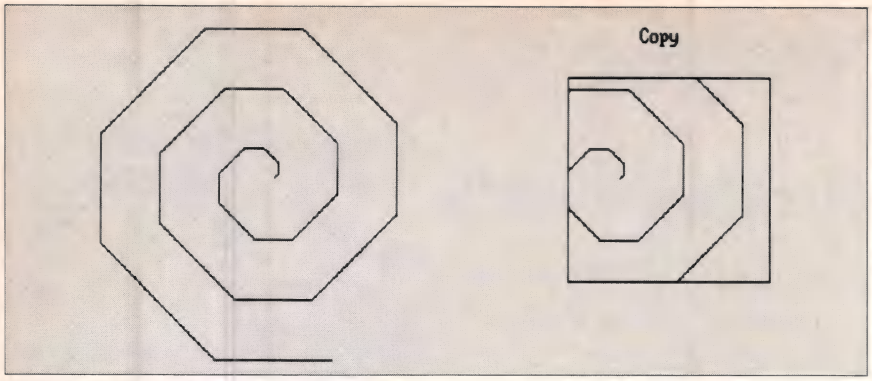


Figure 2: *Output from SPIRAL.MOD (Listing Four)*

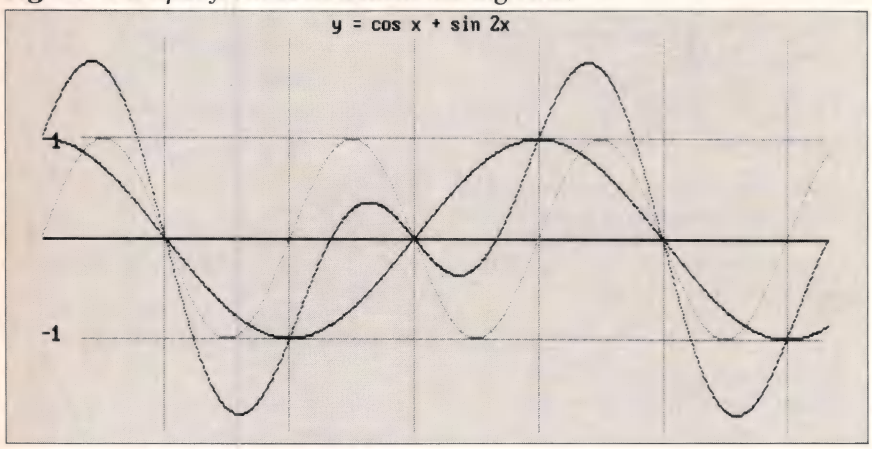


Figure 3: *Photo of MathPlot's output*

(continued from page 121)

dures accept a *color* parameter, but what color applies to *line*? I opted for a "last color used prevails" strategy. The *color* variable, global to the implementation module but not externally visible, forces the next line to inherit the color of the most recently written pixel.

Wirth's approach emulates turtle graphics by employing the concept of a graphics pen that draws a line as it moves, and any activity causes the pen to come to rest at the last pixel written. You can see this idea at work by examining the *line* procedure. *Px* and *Py* are externally visible virtual coordinates giving the pen's position. *Line* advances them before writing each pixel so that, when the line is completed, they reflect the location of the most recent pixel. A pen-up motion is effected by changing the values of *Px* and *Py*; the turtle instantly moves without drawing. The same end results from writing a black pixel (color 3) to a nonsequential location via a call to *dot*.

Why We're Betting a Million Lines of Code on the SAS/C™ Compiler.

At SAS Institute Inc., we've invested more than 10 years of research—and over a million lines of code—in the SAS® System, the world's leading data analysis software. So you can bet we left nothing to chance when we chose the C language for the next generation of our software.

We selected C for the portability it would bring to the SAS System, but weren't about to risk our code on just **any** mainframe C compiler. So we tried them all. When none could meet our exacting requirements, we created our own: the SAS/C compiler.

We Developed It.

Support It. Use It.

The SAS/C compiler set new standards for efficiency and technical quality, with:

- A source-level debugger that includes structure display, ABEND recovery, and debugger I/O exits for debugging specialized applications
- Reentrant object code
- Highly optimized generated code
- Use of standard IBM linkage conventions, with support for 31-bit addressing
- A CMS Rexx/TSO CLIST interface
- Support for signal handling including program checks and terminal interrupts, and non-standard signals such as timer interrupts and stack overflow
- Many built-in functions including string handling
- In-line assembler.

And when we combined these features with outstanding technical support and frequent updates—both provided free—software developers everywhere took notice. The SAS/C compiler is now the market leader, installed in hundreds of commercial firms and academic institutions.

Test It. Compare It.

FREE for 30 Days.

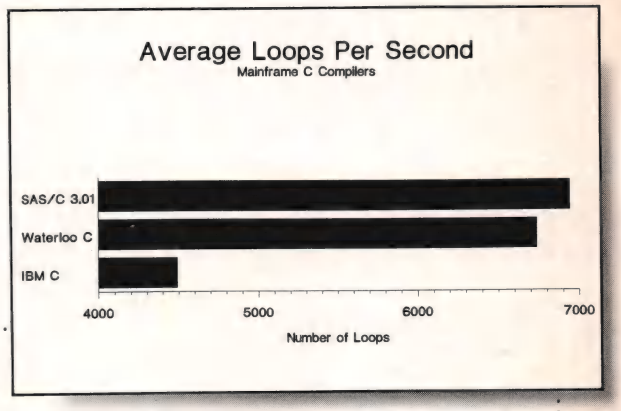
We're betting you've set the same high standards. That's why we'd like to send you the SAS/C compiler, under OS or CMS, for a free 30-day evaluation. We'll also send you a free copy of a leading benchmark program. Compare our compiler with any other. Odds are, you'll choose the SAS/C compiler.

Just mail the coupon below. Or call your Software Sales Representative at (919) 467-8000.

SAS®

SAS Institute Inc.
SAS Circle ☐ Box 8000
Cary, NC 27512-8000
Phone (919) 467-8000
Fax (919) 469-3737

Using a C version of the Dhrystone benchmark, the latest SAS/C compiler release produces the fastest code among the top 3 mainframe compilers. It even tops our own previous release by 35%.



I'd like to put the SAS/C™ compiler to the test with a free 30-day trial, and my free copy of the Dhrystone benchmark program. Give me the details.

Please complete, or attach your business card.

Name _____ Title _____

Company _____

Address _____

City _____ State _____ ZIP _____

Telephone _____

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA, 27512-8000

CIRCLE NO. 196 ON READER SERVICE CARD

DDJ 4/88

(continued from page 122)

The *line* routine in LineDwg is quite limited. It draws in direction *d* for *n* units from the current pen location. The direction indicator *d* is a digit 0..7 expressing a multiple of 45 degrees such that 0 draws to the right, 1 to the northeast, 2 straight up, and so on. To draw lines at unpredictable angles, it's necessary to develop your own routine outside LineDwg; it should call the *dot* procedure for output because *dot* maps virtual coordinates into the physical space.

The *clear* procedure does more than simply clear the screen, but it's consistent with Wirth's ideas. On the IBM PC, any mode change clears the display, even if you switch to the same mode that's currently active. Wirth's demonstration programs inevitably begin with *clear* in lieu of initiating a graphics mode, so the implemented *clear* procedure puts the display into graphics. In the absence of indications otherwise, *clear* also establishes white as the default color. Wirth never calls *clear* again after the start of the run, but your programs can call it any number of times to reset the display without leaving graphics mode.

Using the Module

From the standpoint of the PC, Wirth's programs are inconsiderate because they leave the machine in graphics mode on exit. Obviously he doesn't program on a PC, or he'd have included the switch back to text that precedes the termination of Listing Three, page 90.

This program, called SIERPIN.MOD, is included here as a test to make sure the graphics functions work as expected. Wirth's book shows the output of the program, which is based on a mutually recursive graphics algorithm developed by the Polish mathematician Sierpinski. The only changes from the Wirth model are the addition of the switch back to 80×25-pixel color text at the end and removal of superfluous *Read* statement from the *REPEAT...UNTIL* loop. The output of the program (to my astonishment, it ran correctly the first time) appears in Figure 1, page 118.

The *paint* procedure fills a rectangular area with the specified gray scale. The origin of the area is at (*x*, *y*), extending right for *w* units and up for *h* units, expressed in virtual coordinates. This routine is internally optimized to prevent rewriting pixels that physically coincide as a result of mapping round-offs. Even so, because it's based on ROM BIOS interrupt *10h*, it's agonizingly slow. Much greater efficiency could be achieved by manipulating the EGA registers directly. I chose not to in the interest of limiting the scope of this project; the EGA is a whole big subject of its own.

The *copyArea* procedure also uses the ROM BIOS and is similarly poky in performance, even though—like paint—it's internally optimized to operate at the physical pixel level. This routine copies part of the screen to another location. The source is at (*sx*, *sy*) and the destination at (*dx*, *dy*), the copied area being *dw* virtual units wide by *dh* units high. You'll see it in operation a little later.

The two text routines in *LineDwg* are *Write* and *WriteString*, which parallel similarly named procedures in Modula-2's standard InOut module. The difference is that they start

UNLEASH YOUR 80386!

Your 80386-based PC should run two to three times as fast as your old AT. This speed-up is primarily due to the doubling of the clock speed from 8 to 16 MHz. The new MicroWay products discussed below take advantage of the real power of your 80386, which is actually 4 to 16 times that of the old AT! These new products take advantage of the 32 bit registers and data bus of the 80386 and the Weitek 1167 numeric coprocessor chip set. They include a family of MicroWay 80386 compilers that run in protected mode and numeric coprocessor cards that utilize the Weitek technology.

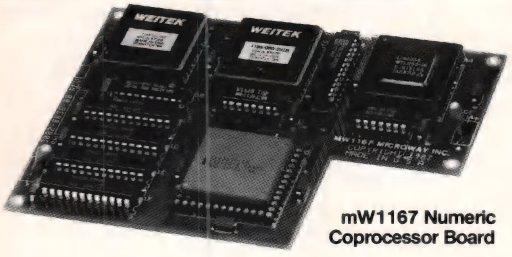
The benefits of our new technologies include:

- An increase in addressable memory from 640K to 4 gigabytes using MS-DOS or Unix.
- A 12 fold increase in the speed of 32 bit integer arithmetic.
- A 4 to 16 fold increase in floating point speed over the 80387/80287 numeric coprocessors.

Equally important, whichever MicroWay product you choose, you can be assured of the same excellent pre- and post-sales support that has made MicroWay the world leader in PC numerics and high performance PC upgrades. For more information, please call the Technical Support Department at

617-746-7341

After July 1988 call 508-746-7341



mW1167 Numeric Coprocessor Board

MicroWay® 80386 Support

MicroWay 80386 Compilers

NDP Fortran-386 and **NDP C-386** are globally optimizing 80386 native code compilers that support a number of Numeric Data Processors, including the 80287, 80387 and mW1167. They generate mainframe quality optimized code and are syntactically and operationally compatible to the Berkeley 4.2 Unix f77 and PCC compilers. MS-DOS specific extensions have been added where necessary to make it easy to port programs written with Microsoft C or Fortran and R/M Fortran.

The compilers are presently available in two formats: Microport Unix 5.3 or MS-DOS as extended by the Phar Lap Tools. MicroWay will port them to other 80386 operating systems such as OS/2 as the need arises and as 80386 versions become available.

The key to addressing more than 640 kbytes is the use of 32-bit integers to address arrays. NDP Fortran-386 generates 32-bit code which executes 3 to 8 times faster than the current generation of 16-bit compilers. There are three elements each of which contributes a factor of 2 to this speed increase: very efficient use of 80386 registers to store 32-bit entities, the use of inline 32-bit arithmetic instead of library calls, and a doubling in the effective utilization of the system data bus.

An example of the benefit of excellent code is a 32-bit matrix multiply. In this benchmark an NDP Fortran-386 program is run against the same program compiled with a 16-bit Fortran. Both programs were run on the same 80386 system. However, the 32-bit code ran 7.5 times faster than the 16-bit code, and 58.5 times faster than the 16-bit code executing on an IBM PC.

NDP FORTRAN-386™ $595
NDP C-386™ $595

MicroWay Numerics

The **mW1167™** is a MicroWay designed high speed numeric coprocessor that works with the 80386. It plugs into a 121 pin "Weitek" socket that is actually a super set of the 80387. This socket is available on a number of motherboards and accelerators including the AT&T 6386, Tandy 4000, Compaq 386/20, Hewlett Packard RS/20 and MicroWay Number Smasher 386. It combines the 64-bit Weitek 1163/64 floating point multiplier/adder with a Weitek/Intel designed "glue chip". The mW1167™ runs at 3.6 MegaWhetstones (compiled with NDP Fortran-386) which is a factor of 16 faster than an AT and 2 to 4 times faster than an 80387.

mW1167 16 MHz $1495
mW1167 20 MHz $1995

Monoputer™ - The INMOS T800-20 Transputer is a 32-bit computer on a chip that features a built-in floating point coprocessor. The T800 can be used to build arbitrarily large parallel processing machines. The Monoputer comes with either the 20 MHz T800 or the T414 (a T800 without the NDP) and includes 2 megabytes of processor memory. Transputer language support from MicroWay includes Occam, C, Fortran, Pascal and Prolog.

Monoputer T414-20 with 2 meg[1] ...$1495
Monoputer T800-20 with 2 meg[1] ...$1995

Quadputer™ can be purchased with 2, 3 or 4 transputers each of which has 1 or 4 megabytes of memory. Quadputers can be cabled together to build arbitrarily fast parallel processing systems that are as fast or faster than today's mainframes. A single T800 is as fast as an 80386/mW1167 combination!

Biputer™ T800/T414 with 2 meg[1] $3495
Quadputer 4 T414-20 with 4 meg[1] ...$6000

[1] Includes Occam

80386 Multi-User Solutions

AT8™ - This intelligent serial controller series is designed to handle 4 to 16 users in a Xenix or Unix environment with as little as 3% degradation in speed. It has been tested and approved by Compaq, Intel, NCR, Zenith, and the Department of Defense for use in high performance 80286 and 80386 Xenix or Unix based multi-user systems.

AT4 - 4 users $795
AT8 - 8 users $995
AT16 - 16 users $1295

Phar Lap™ created the first tools that make it possible to develop 80386 applications which run under MS-DOS yet take advantage of the full power of the 80386. These include an 80386 monitor/loader that runs the 80386 in protected linear address mode, an assembler, linker and debugger. These tools are required for the MS-DOS version of the MicroWay NDP Compilers.

Phar Lap Tools $495

PC/AT ACCELERATORS

287Turbo-10 10 MHz $450
287Turbo-12 12 MHz $550
287TurboPlus-12 12 MHz $629
FASTCACHE-286 9 MHz $299
FASTCACHE-286 12 MHz $399
SUPERCACHE-286 .....$499

MATH COPROCESSORS

80387-20 20 MHz $895
80387-16 16 MHz $495
80287-10 10 MHz $349
80287-8 8 MHz $259
80287-6 6 MHz $179
8087-2 8 MHz $154
8087 5 MHz $99

MicroWay

The World Leader in PC Numerics

P.O. Box 79, Kingston, Mass. 02364 USA (617) 746-7341
32 High St., Kingston-Upon-Thames, U.K., 01-541-5466
St. Leonards, NSW, Australia 02-439-8400

writing at the current pen position—well, near it, anyway. This implementation uses the default EGA character set rather than a stroked font, so it places the text cursor—hence the first character—in the 8×14 physical cell occupied by the pen. That's as close as you can get with the built-in characters, and it should be good enough for most applications. The foundation routine is *Write*, which outputs a single character and advances the pen. *WriteString* merely calls *Write* for each character in the string.

This is where the bug in Stony Brook Modula-2 popped up. It's unclear what's wrong, but literals passed as parameters to *WriteString* sometimes show up with a garbage character appended to the end. Either the compiler isn't placing a null terminator in literals, or else the *Length* procedure has a glitch. Through trial and error, I discovered that an extra space at the end of the literal makes the problem go away sometimes, and other times taking that padding space out has the same effect. This is a bug in the category of an annoyance and hardly a major flaw.

The SPIRAL.MOD program in Listing Four, page 92, exercises all directions of *line*, plus *copyArea* and the text-writing routines. Though a small program, it runs for quite a while because of the inefficiency of ROM BIOS calls, two of which are made for each pixel copied to the outlined window at the right side of the screen. The figure copied is a portion of the spiral pattern; Figure 2, page 122, shows the final screen.

These two demonstration programs are nice to look at, especially the interesting Sierpinski curve, but they have no practical value. For that reason, I've thrown in MATHPLOT.MOD (Listing Five, page 93) to illustrate a more useful application of Wirth's LineDrawing module. This program plots the function y = cos x + sin 2x, which is representative of any number of mathematical graphing applications in both business and scientific programming.

MathPlot draws three curves in different gray scales using the *dot*

routine. Dark gray follows sin 2x, light gray plots cos x, and white is the curve deriving from their sum, where $0 < x < 10$. Thus the curves go slightly beyond $x = 3\pi$, tracing the function through a bit more than 1.5 circles. The vertical grid lines delineate intervals of 2π, and the horizontal grids indicate unity on either side of the x axis, giving visual reference points for the curves. The photo in Figure 3, page 122, doesn't do it justice. This is a handsome display that you have to see on an EGA screen to appreciate the usefulness and elegance of the gray-scale plots produced by Wirth's LineDrawing module.

Stony Brook's Modula-2

This experience with Stony Brook's Modula-2 sold me on the product. Guys like me, writing for several computer magazines and on a first-name basis with the Fed Ex man because he brings so many review copies, tend to get jaded. It's hard to work up enthusiasm over yet another new product in an industry that sprouts them like weeds on a vacant lot. So praise doesn't come easily.

I'm not saying it's perfect. In addition to the bug I mentioned earlier, I found a truly ugly glitch while developing MathPlot. In the *yc* procedure, I originally wrote *INTEGER (y * 100)*. This is a dumb mistake in that a type transfer (or cast in C parlance) isn't a true function and thus doesn't resolve parametric expressions. I just forgot, and I'm not ashamed; programmers make such mistakes all the time. But it gave the compiler a nervous breakdown, causing it to generate a screenful of cryptic internal error messages. I spent half an hour commenting out program lines one by one before discovering the offender. And in fixing that problem, I found an error on page 114 of the manual, which says that the *entier* function (*REAL* to *INTEGER*) returns a *REAL*, which is clearly wrong. There are probably a few others of this sort that I haven't uncovered yet.

But overall, Stony Brook Modula-2 is good. It doesn't have quite as many bells and whistles as its better-known competitor from Logitech, but the tools it provides are well rounded: 23 built-in modules, an editor, a MAKE utility, and a symbolic debugger. It supports the usual array of memory models, has a satisfying selection of compiler options, and uses the standard DOS linker. Above all, it's fast. If you program in Modula-2, a viable and less intimidating alternative to C as a systems programming language, you'll like it. I sure do.

Do you have a programming problem you'd like to see addressed here? If so, drop me a line at *DDJ* (no calls, please!) or leave a message for KPORTER, Mountain View, CA, on MCI Mail. No promises, but all suggestions are welcome; dreaming up a subject for a monthly column is a fast track to burn-out.

DDJ

(Listings begin on page 88.)

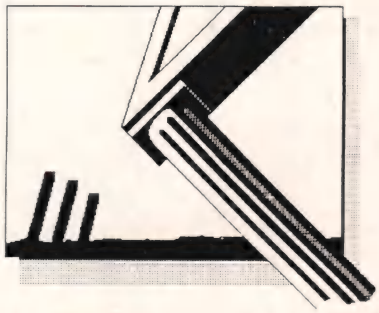
Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

The Advertiser Index

*The advertiser prefers to be contacted by phone; consult ad.

THE FORTH COLUMN

by Martin Tracy



The Ninth Forth Modification Laboratory (FORML) Conference was well attended and well worth going to. It was held on a beautiful Thanksgiving weekend in Monterey, Calif., and you could watch the pelicans migrating southward along the Pacific Coast and see swarms of Monarch butterflies hanging from the trees.

As usual, there were two days of solidly packed presentations, with wine and cheese in the evenings. Stephen Sjolander presented a Novix 4016 decompiler that would be of great value to anyone working with this chip. Bob La Quey pointed out that Forth would be an excellent language for implementing Hypertext. George Shaw presented the most complete and comprehensible paper on QUANs (multiple code field words) that I have ever seen. Stephen Pelc showed an implementation of record and field defining words, and Dr. Ting shared a short recursive line-drawing algorithm. This year, Dr. Ting presented even more papers (four) than Wil Baden (three)!

There were many other high-quality papers that you will be able to read in the *1987 FORML Conference Proceedings*, available from the Forth Interest Group ([408] 277-0668) later this year. Speaking of that, the *Proceedings of the 1987 Rochester Forth Conference* is now available as *The Journal of Forth Application and Research* (*JFAR*), vol. 5, no. 1, 1987, from the Institute of Applied Forth Research, 70 Elmwood Ave., Rochester, NY 14611. Or you can order it indirectly from FIG.

The informal theme of FORML this year seemed to be *BLOCK*s vs. text files. Each attendee received a copy of Tom Zimmer's PF Forth (formerly ZF Forth and soon to be FF Forth). Tom's IBM PC public-domain dialect is the joint effort of Zimmer, Smith, Curley, and Modrow. It is enormous by Forth standards, occupying 1 Mbyte of disk space. It includes source files for dozens of utilities. You can get an ARCed copy from the GENie Forth bulletin board (see *DDJ*, December 1987). Expect to spend a lot of weekends learning it.

PF Forth keeps all its source in text files and includes a sophisticated file editor. The program image is in a .EXE file, and headers are kept in a separate segment. It is closely related to Mike Perry's unpublished F83Y. Mike is the Perry behind the Laxen/Perry F83 dialect. (Henry Laxen has, for the moment, died and gone to C.)

In Mike's own words: "In an additional attempt to merge with the environment, explore the trade-offs, and generally ingratiate myself with certain fanatic elements, I discarded BLOCK. Don't worry, it's still available as a loadable option. When a file is loaded, it is read into the current file segment with a single call to DOS, then interpreted. Note the 64K limit. All control characters and blanks are treated as white-space. This allows a single compare to be used. Compilation is quite fast compared to whatever I used to use."

The first ever Australian Forth Symposium will be held May 19-20, 1988, at the NSW Institute of Technology. The focus is on productivity, and there will be papers, demonstrations, and hands-on instruction. Chuck Moore will be there. If you want to go, ring Jose Alfonso or Dr. Walker on (02)20930 or Roy Hill on (02)217-3828.

The next ANSI X3J14 Forth standards meeting is also in May, somewhere in the Northeastern United States. Contact me for details if you would like to attend. You can reach me at Forth Inc., 111 N. Sepulveda Blvd., Manhattan Beach, CA 90066.

At the second ANSI Forth meeting (November 1987), preparations were made for accepting technical proposals. A copy of the approved proposal form is on page 132. About two dozen proposals have been received and many more are sure to follow. There seem to be roughly a dozen technical issues: vocabularies, mass storage, flow of control, arithmetic, documentation requirements, testing, assemblers, controlled reference word set, ROMability, host and file structure, the interpreter, Tick >BODY and addressability, numeric output, and exception handling.

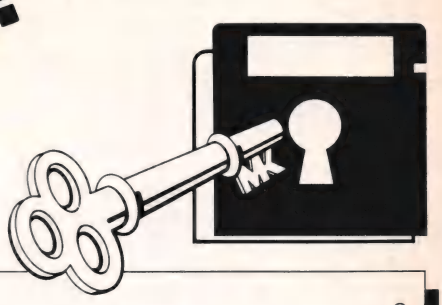
Each member of the ANSI Forth committee will prepare a position paragraph on each issue. One member, the "magnet," will collect all the paragraphs and synthesize them into one paper describing the different sides of the issue. You can follow or participate in the discussion on any issue by signing onto the GENie Forth bulletin board, Category 10, Forth Standards. I'm told that several hundred people have already perused this category.

Two New Books on Forth

You will find the new *Dr. Dobb's Toolbook of Forth, Volume II*, to be an excellent anthology of papers on Forth programming techniques. Inside, you can find Bresenham's algorithm, FFTs, spreadsheets, and Logo, all implemented in Forth. Some papers are taken from FORML and Rochester proceedings, and others first appeared here in *DDJ*. Unfortunately, credit is not given to the

MASTER*KEY

Unlocks Everything!



turn this into this!

```
C:>DEBUG PROGRAM.COM
-D100 136
8848:0100  EB 18 49 6E 63 6F 72 72-65 63 74 20 44 4F 53 20  k.Incorrect DOS
8848:0110  76 65 72 73 69 6F 6E 0D-0A 24 50 B4 30 CD 21 86  version..$P40M!.
8848:0120  E0 3D 36 01 72 05 3D 0A-02 76 09 BA 02 01 B4 09  `=6.r.=..v.:..4.
8848:0130  CD 21 CD 20 58 EB 2F                             M!M Xk/
-G
```

MASTER*KEY
No Other Product Comes Close!

An EXPERT may not know the solution, but always knows where to find it.

MASTER*KEY HELPS ANYONE solve those confusing and frustrating software puzzles more rapidly and easily than any other software available, at any cost! It gives you know-how within hours that may otherwise take years of experience. Create a new program from an old one. DON'T REINVENT THE WHEEL!

MASTER*KEY - Smart!

MASTER*KEY is an intelligent self-documenting MS-DOS reverse assembler. Its sophisticated procedures swiftly race through massive and baffling object code files to effortlessly discover potential trouble spots.

MASTER*KEY - Educational!

YOU DON'T NEED TO KNOW ASSEMBLY LANGUAGE! MASTER*KEY will take any program from your IBM-compatible computer and return fully-documented, easily-understood assembly language source code (Microsoft MASM 5.0 compatible).

MASTER*KEY - Easy To Use!

MASTER*KEY works both automatically from the DOS command line or interactively from menus similar to Lotus Corporation's 1-2-3 or Symphony. No need to remember any new commands or continually refer to a manual. Use it immediately!

Minimum System Requirements:
256K + 8088/8086/80186/80286/80386 PC
MS-DOS or PC-DOS 2.0 +
One 360K DSDD Floppy Drive (IBM PC Format)

MS-DOS is a trademark of Microsoft.
PC-DOS is a trademark of IBM.

```
H00100: JMP     Short H0011A                    ;00100 EB18          __
;-----------------------------------------------
        DB      "Incorrect DOS version"         ;00102 496E636F727265
        DB      0Dh                             ;00117
        DB      0Ah                             ;00118
        DB      "$"                             ;00119 24
;-----------------------------------------------
H0011A: PUSH    AX                              ;0011A 50            P
        MOV     AH,30h                          ;0011B B430          _0
        INT     21h             ;1-DOS_Ver_Number ;0011D CD21        _!
        XCHG    AH,AL                           ;0011F 86E0          __
        CMP     AX,0136h                        ;00121 3D3601        =6_
        JB      H0012B          ; . . . . . .   ;00124 7205          r_
        CMP     AX,020Ah                        ;00126 3D0A02        =__
        JBE     H00134          ; . . . . . .   ;00129 7609          v_
H0012B: MOV     DX,0102h                        ;0012B BA0201        ___
        MOV     AH,09h                          ;0012E B409          __
        INT     21h             ;1-Display_String ;00130 CD21        _!
        INT     20h             ;TERM_normally:20h ;00132 CD20       _
;-----------------------------------------------
H00134: POP     AX                              ;00134 58            X
        JMP     Short H00166                    ;00135 EB2F          _/
;-----------------------------------------------


MASTER*KEY XREF - PROGRAM.XRF                                  Page 1

0102h                   :   121   2F5   301   320
020Ah                   :   126
03CBh                   :   12B
1-Display_String        :   130   591   610
1-DOS_Ver_Number        :   11D
H00100                  :   100
H0011A                  :   100   11A
H0012B                  :   124   12B
H00134                  :   129   134
H00166                  :   135
TERM_normally:20h       :   132
```

NOTE: The cross-reference is by memory location within the program file!

NOTE: The output is totally Microsoft MASM-compatible.

(not copy protected)

MASTER*KEY will guide you step by step to:

1. Help you learn assembly language, if you desire.
2. Discover how any program runs or why it doesn't.
3. Alter or remove unwanted object code from any program.
4. Incorporate routines from compiled programs into other assembly language, Basic, C, or Pascal programs.
5. Make software more compatible with your computer. Be certain a questionable program won't damage your system BEFORE you run it.
6. Modify software to operate with other versions of DOS.
7. Customize your COMMAND.COM or other executable program directly or by reassembling your altered MASTER*KEY source code.

Order Now! Just $79.95

Phone orders accepted on MC or VISA
$82.45 (includes $2.50 shipping)
$87.65 in California (includes tax & shipping) **C.O.D.** orders add $2.00

(714) 596-0070

Dealer/Distributor Inquiries Welcome

Sharpe Systems Corporation
2320 E Street, Dept. 44, La Verne, CA 91750 714-596-0070

MASTER*KEY should not be confused with any public domain or share ware software that may have a similar name or be a similar product.

Please send **MASTER*KEY!**

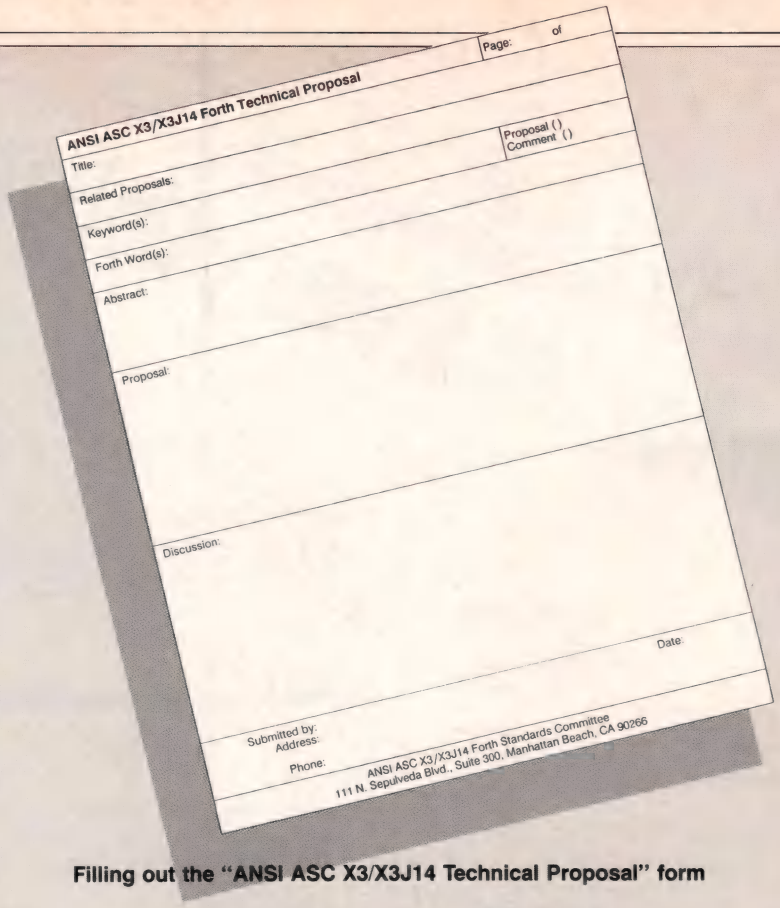
Send checks to:
Sharpe Systems Corporation
2320 E Street, Dept. 44, La Verne, CA 91750

Name _____

Address _____

City _____ State _____ Zip _____

CIRCLE NO. 203 ON READER SERVICE CARD



Filling out the "ANSI ASC X3/X3J14 Technical Proposal" form

Page:	Please number each page of your proprosal.
Title:	A short phrase that characterizes your proposal. Example: String extensions.
Related Proposals:	If you have submitted other proposals on this subject, please list their titles and dates.
Proposal( )	Are you proposing a specific change? Then check the Proposal box.
Comment( )	Otherwise, check the comment box.
Keyword(s):	Pick a keyword that helps others understand what area of Forth your proposal addresses. For example: Kernal, double integers, system word set. . .
Forth word(s):	List all affected Forth words, including ones added or deleted by your proposal or discussed in your comments.
Abstract:	Briefly convey the nature of your proposal (or comment).
Proposal:	State your proposal in specific terms. When proposing a new word, or changes to an existing word, simply state the new definition. Include a stack picture if appropriate.
Discussion:	If you are making a comment, put it here. If you are submitting a proposal, provide compelling arguments in favor of your proposal. Be concise. If you are aware of arguments against your proposal, state them and rebut them. Is your proposal consistent with current Forth conventions? Does the proposal involve hardware or other system dependancies? Does if affect application portability? Does it have implications for multi-user environments? Should the proposed change by mandatory? Does it affect ROMability? Is it general purpose? How does it affect execution speed? How does it affect compilation speed? What are its memory requirements?
Submitted by:	Provide your name, address, and daytime phone number.

Use the "ANSI ASC X3/X3J14 Technical Proposal Form, cont'd" form if additional pages are needed. Remember to put page numbers and repeat the "Submitted by" and "Date" fields on each page.

Figure 1: *Approved ANSI Forth proposal form*

THE FORTH COLUMN
(continued from page 130)

original publication. There are 32 quality papers in all, many of them with source code. The Toolbook sells for $29.95 and is available from *DDJ* ([800] 533-4372 or [800] 356-2002 in California). If you don't want to type in the programs, you can get them on a disk for an additional $25.

Dick Pountain's *Object-Oriented Forth* (Academic Press, 1987; $19.95) begins: "Forth is, regrettably, one of the best kept secrets in the computing world." In this book, Dick extends Forth to include named objects, which increases its readability and reusability. In essence, an object is a defining word used to create defined words called instances. An instance has its own hidden data structure, made of instance variables, and a set of permitted operations, called methods. Instance variables are to instances as user variables are to tasks.

When an object is defined, all instance variable and method names are removed from the Forth dictionary by relinking the headers. Only the object itself can find them because it has the initial link, or "key," to the hidden vocabulary. All instances of this object are state smart and know the key. During compilation an instance looks the following method up in its private dictionary and compiles a reference to it.

As an example, consider complex numbers as objects. They might have instance variables for the real and imaginary parts and methods for addition, multiplication, and so on. Pountain limits his methods to @ and !, however, so that complex results are left where they belong, on the stack:

```
TYPE> COMPLEX
   2 VAR REAL
   2 VAR IMAG
OPS>
   : COM! IMAG ! REAL ! ;
   : COM@ REAL @ IMAG @ ;
ENDTYPE> COMPLEX

COMPLEX X 2 3 X COM!
```

Object-Oriented Forth (*OOF?*) takes you through this and other abstract data types, including lists

(continued from page 133)

and heaps. This book is not for beginners, and I highly recommend it.

Dead Duck's Hard Beak

Starting in the October 26, 1987, issue of *InfoWorld*, Steve Gibson ran three articles in his Tech Talk column describing the implementation of Microsoft's QuickBASIC 4.0. According to Mr. Gibson, "The technology is known as 'Threaded PseudoCode' ... and was first seen in the strange powerful language Forth." He continues, "Forth is the language you want to pretend doesn't exist because it's horrible to write with but hauntingly powerful."

Needless to say, these comments drew immediate rebuttal from the Forth community. Here is a sample, translated from Timothy Huang's Chinese Forth column: "... the term Threaded PseudoCode was used repeatedly with such an envious attitude, like a human being who just discovered the warmth of the sun. In fact, we have been using this technique for more than ten years. So why doesn't Mr. Gibson and Microsoft, as well as the whole computer industry, come straight out and recognize FORTH, and quit behaving like a dead duck with a very tough beak?"

Text from BLOCKs

In my last column, February 1988, I presented command for reading and writing data and text files from *BLOCK*s. Some of the source screens needed fixing, and you will find corrected versions in this month's Listing One, page 94.

The new *GETTEXT* includes the flow-of-control construct *BEGIN... WHILE...WHILE..REPEAT*. This construct occurs naturally in both string and file applications:

```
BEGIN ( Are there more characters?)
WHILE ( Is this one that you want?)
WHILE ( Go ahead and use it.)
REPEAT
```

Note that this is a proper structure, delimited by *BEGIN* at one end and *REPEAT* at the other.

In most Forths, the stack is used at compile time to hold fix-up addresses until they can be resolved. In some dialects, such as polyFORTH and ZEN, only these addresses are on the stack. Furthermore, *WHILE* swaps addresses to keep the *BEGIN* address on top. *BEGIN...WHILE...WHILE..REPEAT* can be implemented simply by adding *THEN* to resolve the second *WHILE*, as in *BEGIN... WHILE... WHILE...REPEAT THEN*.

Other Forths, especially those descending from FIG Forth, keep extra flags on the stack for "compiler security." If *BEGIN...WHILE...WHILE ... REPEAT* does not work for your Forth, you can change it to the less elegant construct:

```
BEGIN ( Any more characters?)
DUP IF DROP ( Want this one?)
THEN
WHILE ( Eureka! Use it.)
REPEAT
```

DDJ

(Listing begins on page 94.)

Quit Wasting Time!

As a programmer, most of your time is spent writing and debugging source code, and documenting your work. A powerful, easy-to-use programmable text editor could be saving you HOURS of unnecessary effort.

Only MULTI-EDIT has all these time-saving features:

Fully automatic Windowing and Virtual Memory.
Edit multiple files regardless of physical memory size.
Easy cut-and-paste between files.
View different parts of the same file.

Powerful, EASY-TO-READ high-level macro language.
Standard language syntax.
Full access to ALL Editor functions.
Automate repetitive tasks.
Easy, automatic recording of keystrokes.

Language-specific macros for ALL major languages.
Smart indenting.
Smart brace/parenthesis/block checking.
Template editing.
Supports C, Pascal, BASIC and Assembler.

Terrific word-processing features for all your documentation needs.
Intelligent word-wrap.
Automatic pagination.
Full print formatting with justification, bold type, underlining and centering.
Even a table of contents generator.

Compile within the editor.
Automatically positions cursor at errors.
Built-in MAKE capabilities.
Run compiled program without leaving editor.
Automatically allocates all available memory to compiler or program.

Complete DOS Shell.
Scrollable directory listing.
Copy, Delete and Load multiple files with one command.
Background file printing.

Regular expression search and translate.
Condensed Mode display, for easy viewing of your program structure.
Pop-up FULL-FUNCTION Programmer's Calculator and ASCII chart.

and MOST IMPORTANT,
the BEST user-interface on the market!

- Extensive context-sensitive help.
- Choice of full menu system or logical function key layout.
- Function keys are always labeled on screen (no guessing required!).
- Excellent online, interactive tutorial.
- Keyboard may be easily reconfigured and re-labeled.

Users of Wordstar and Turbo Pascal's Editor could be programming in a fraction of the time with these features.

NO EDITOR ON THE MARKET TODAY HAS ALL THESE FEATURES, OR OFFERS YOU THIS MUCH POWER AT A REASONABLE PRICE, EXCEPT

Multi-Edit VERSION 2.0 $99.

COMPLETE
Or Get our FULLY FUNCTIONAL DEMO Copy for only $10!

	Multi-Edit	BRIEF 2.0	Norton Editor	Vedit Plus
Edit 20+ files larger than memory	Yes	Yes	No	Yes
Powerful high level macro language	Yes	Yes	No	Yes
Full UNDO	Yes	Yes	No	No
Visual marking of blocks	Yes	Yes	Yes	No
Column oriented block operations	Yes	Yes	No	No
Automatic file save	Yes	Yes	No	No
Online help	Extensive	Limited	Limited	Limited
Online tutorial	Yes	No	No	Yes
Choice of keystroke commands or menu system	Yes	No	No	Yes
Function Key assignments labeled on screen	Yes	No	No	No
WP Functions	Extensive	Limited	Limited	Extra Cost
Complete DOS shell	Yes	No	No	No
Pop-up Programmer's Calculator and ASCII table	Yes	No	No	No ASCII
Unlimited 'Off the Cuff' keystroke macros	Yes	No	No	Yes
Allocates all available memory to compiler when run from within editor	Yes	No	Yes	Yes
Intelligent indenting, template editing and brace/parenthesis/block matching and checking for all major languages	Yes	C Only	No	Limited
Flexible condensed mode display	Yes	No	Yes	No
Optional background communications and Spell Checker modules	Yes	No	No	No
PRICE	**$99**	**$195**	**$50**	**$185**

Requires IBM/PC/XT/AT/PS2 or full compatible, 256K RAM, PC/MS-DOS 2.0 or later-Multi-Edit and American Cybernetics are trademarks of American Cybernetics. BRIEF is a trademark of Underware, Inc. Norton Editor is a trademark of Peter Norton Computing, Inc. Vedit is a registered trademark of CompuView Products Inc. Copyright 1987 by American Cybernetics.

Get our FULLY FUNCTIONAL DEMO Copy for only $10!

To Order, Call 24 hours a day:
1-800-221-9280 Ext. 951
In Arizona: 1-602-890-1166
Credit Card and COD orders accepted

American Cybernetics
138 Madrid Plaza
Mesa, AZ 85201

CIRCLE NO. 207 ON READER SERVICE CARD

EXAMINING ROOM

Turbo Professional 4.0

Product:
Turbo Professional 4.0
Target:
PC or PS/2 and compatibles
Requires:
DOS 2.0 or later; Turbo Pascal 4.0 or later; hard disk recommended
Pricing:
$99
Vendor:
Turbo Power Software, 3109 Scotts Valley Dr., Ste. 122
Scotts Valley, CA 95066; (408) 438-8608

If you want to protect your reputation as a guru, don't let anybody know you use the Turbo Professional 4.0 library. The guys down at Turbo Power Software have made the hard stuff so easy that it's bound to reduce your fellow programmers' awe of your coding prowess.

Imagine! You can create a pop-up window with a one-liner, and with other one-liners you can rearrange a stack of windows to bring the selected one to the top, make any of them go away, and write to the active window. Or you can similarly manipulate EMS and pull-down menus, insert chained interrupt handlers, and manage TSRs.

All kidding aside, Turbo Professional 4.0 is a superbly crafted toolbox for serious programmers using Turbo Pascal 4.0. Consisting of some 300 functions and procedures spread across 25 Pascal units, it truly does render the difficult accessible. Some of these routines address defects in Turbo Pascal 4.0—lack of a BCD type, for example—while others simplify the dirty work of advanced applications development. My particular favorite, the unit on TSR management, removes the mysteries and perils from this most frustrating kind of program. There are even functions for changing the hot key on the fly and disabling a TSR. The manual has six lucid pages of discussion that make TSRs make sense. The 426-page manual does a workmanlike job of explaining how all these abstractions of complexity fit together and operate, and what the parameters mean.

There isn't room here to cover all the goodies in this package, so I'll list the major categories and discuss a few particulars. These are the groupings of routines:

- Screen: Virtual screens, popups, pulldowns, input editing
- Strings: Manipulations, formatting, long ASCIIZ, DOS command line parser
- Low-level: Sugar-coated DOS and ROM BIOS calls
- Interrupts and TSRs: ISR and TSR management, 8087 TSR support, critical error handler
- Sorting: In-memory non-recursive quicksort of up to 64K elements
- Memory: EMS and extended memory management
- Macros: Keyboard macros and an editor
- Large arrays: In RAM, EMS, and virtual memory
- BCD: 18-digit precision for arithmetic and transcendentals
- Other: Runtime error recovery, inline macros

Additionally, the package includes about a dozen demo programs that really are useful, such as a programmer's calculator (a TSR written with the library), time management/billing utilities, and a GREP-like text finder.

Turbo Professional comes on three diskettes, only one of which (the TPU disk) you need to copy to your Turbo Pascal 4.0 directory in order to use the software. The other two disks contain the demo programs and archived source code for the entire library, along with an unarchiving utility.

You harness the horsepower of this programming engine with the USES statement. All the units have the name pattern TP*.TPU, which makes them easy to identify. There are some unit dependencies, and several depend on Pascal's DOS unit as well. Appendix A in the manual lists them. As an example, to do virtual screens, the directive is

```
USES DOS, TPCRT, TPScreen;
```

The syntax of some of the calls is a little awkward, but that doesn't diminish their power. For example, to initialize and work with a pop-up window, the call is

```
IF NOT MakeWindow (win1, . . .)
THEN
  {Error occurred}
ELSE
  {Do window stuff}
```

(MakeWindow is a function taking 12 arguments.) If MakeWindow is successful, you can pop it up with

```
BoolResult := SetTopWindow
(win1);
```

where BoolResult becomes TRUE if the operation works. Getting rid of the most recently popped window is a different matter:

```
winPtr := EraseTopWindow;
```

Ron Copeland, associate editor, is the coordinator for this review section. He welcomes your feedback on products worth reviewing.

If you ever wanted to take a crack at assembly language, now's the time.

You probably already know that assembly language subroutines are the smartest way to get the fastest programs.

But if the complexities of working in assembler made you think twice, here's some good news. We've made Microsoft® Macro Assembler Version 5.0 a lot easier to use.

We eased the learning process by giving you the best support around. We completely revised our documentation. The new Mixed Language Programming Guide gives you step by step instructions for linking your assembly code with Microsoft QuickBASIC, C, FORTRAN, Pascal and other languages. And you get a comprehensive reference manual with listings of the instruction set and examples of each instruction. We didn't stop there, though. You also get an on-disk collection of templates and examples.

We've also dramatically simplified the high-level language interface. In just a few simple steps, you can be calling Macro Assembler subroutines from programs written in your favorite language.

Now that you're writing the fastest programs, Microsoft is giving you the fastest way to debug them. For the first time, we've added our CodeView® debugger to Macro Assembler. With source code and comments on your screen, Microsoft CodeView makes debugging programs containing assembly language subroutines a snap.

And you'll be glad to know that you don't sacrifice any speed for all the ease of use.

We took the fastest Macro Assembler on the market and made it even faster.

So what are you waiting for? Get your hands on Microsoft Macro Assembler and see what it's like to break your personal speed limit.

Microsoft®

For more information or for the name of your nearest Microsoft dealer, call (800) 426-9400. In Washington State and Alaska, (206) 882-8088. In Canada, call (416) 673-7638.

(continued from page 136)

which returns a pointer to the window erased. The extensive use of such value-returning functions rather than more intuitive procedures gives the library a C-like feel.

The BCD module brings 18-digit precision to Pascal. This is much better than the 11-maybe precision of the REAL type and not subject to roundoff errors or the unexpected appearance of E notation. BCD is usually associated with business programming, but Turbo Professional makes it viable for scientific applications as well by including trig and exponential functions.

Inclusion of the source code with Turbo Professional is a nice touch. If you think certain routines ought to work differently, you can change them. Some are written in Pascal, but most are in assembler, so you'll need MASM or something like it if you plan to do much rewriting. Of course you can also add functionality to suit.

I have only one complaint about Turbo Professional. I never did figure out how to work the Menu Maker utility, and the documentation didn't help. That problem aside, for serious developers who work in Turbo Pascal 4.0, Turbo Professional is an outstanding productivity aid that will pay for itself many times over.

by Kent Porter

Brief

Product:
Brief, Version 2.01
Target:
PC or PS/2 and compatibles
Requires:
DOS 2.0 or later; 192K minimum
Pricing:
$195
Vendor:
Solution Systems
541 Main St., Ste. 410
South Weymouth, MA 02190
(617) 337-6963

One might fairly describe Solution Systems' Brief as the quintessential programmer's editor. Unfortunately, that doesn't tell you much. So maybe I'd better tell you why I think it's so terrific.

To begin with, it's chock full of the features that programmers routinely require. And in the event that you find it wanting in a specific function or feature, Brief has its own C-like language for writing macros or otherwise tailoring it to your requirements.

Although Brief isn't copy protected, it's necessary to run an installation program to configure it. There are scads of options: colors, cursor shapes, tab settings, 25– versus 43– versus 50–line mode, backup files, and what not. This process uncovers some intriguing features. For example, you can set up Brief to save your work automatically every so many seconds or minutes, thus minimizing the impact of a power failure. For another, you can establish different parameters for each language you use.

Brief does all the usual editing things: line insert/delete, search and replace, block move/copy, etc. Most editing commands are accomplished by ALT-key sequences.

Brief gives you plenty of room to work in; maximum size of a text buffer is 32 Mbyte (DOS limit), with the maximum number of buffers limited only by available RAM. Macro size is also plentiful with a 32K limit and the maximum number of macros limited by available RAM.

A particularly nice feature is the Undo command, ALT-U. Since Brief maintains an internal change stack, by default, this stack keeps track of the most recent 30 changes. And if you wish, you can increase it as high as 300. So, if you find that some program modification is headed down the primrose path, just start hitting ALT-U to back out the changes you've made. Brief "undoes" them in LIFO order.

The real stuff of Brief is its multifile and windowing capabilities. The program comes up in full-screen edit mode, where one file owns the entire screen. You can load any number of files and move among them either in carousel fashion or via direct jumps using ALT commands, with each in full-screen mode. Alternatively, you can split the screen into resizable tiled windows and put a different file in each one, or different parts of the same file in two or more windows. If several windows contain the same source code, a change in one immediately appears in all.

The import of this is that if two modules are related, you can look at both on the same screen, jumping between to reconcile their differences. Or, in a single program, you can display a structure definition in one window while you write its initialization code in another, which displays a different part of the file. In a complex application, this feature alone can save an astronomical amount of time simply by eliminating the need to save and load files, scroll around, and print listings that will immediately become out of date.

Also, Brief's cut-and-paste and copy operations span multiple buffers using a clipboard concept. This lets you take a function out of PROG1 and insert it into PROG2 with half a dozen keystrokes.

On the negative side, moving among windows is awkward. You press the F1 key, then a cursor arrow indicating which direction you want to go. You have to repeat this in every window you pass through between origin and destination. It would have been better if they'd numbered the windows so you could do F1-window number.

Brief lets you set up and select language-specific environments. For example, indentation conventions are different for Assembler than for block-structured languages such as Pascal, C, and Modula-2, as is case sensitivity (Pascal and Assembler aren't, C and Modula are). The default filename extensions also differ. The setup program builds parameter sets for each language. In it you can also specify the compiler command line, enabling you to invoke the compiler from within Brief. This effects a Turbo/Quick-style "environment," although of course there isn't the same degree of interaction between editor and compiler that characterize true integrated environ-

C the Improvement.

NOW VERSION 3.3 For MS-DOS and OS/2!

Now the Lattice C Compiler takes you where it's never gone before. With Version 3.3, it works on *two* operating systems: MS-DOS *and* OS/2!

You may now use Version 3.3 on an MS-DOS system to create programs that run under OS/2 protected mode. Or vice versa. A simple "switch" has been put into the compiler to let you generate code for either system or both.

New improved standards...

Version 3.3 is fully compliant with the latest ANSI C standards. It also has improved embedded system support, enhancements to the standard libraries and a host of other compiler advances too numerous to compile.

At a new improved price and value!

The suggested retail for Lattice C Version 3.3 is only $450. And 3.3 also includes "family" versions of the Lattice Screen Editor (LSE) and the Lattice C-SPRITE™ symbolic debugger, compatible with both MS-DOS and OS/2 systems, at no charge.

C for yourself why Lattice is the professional programmer's choice for serious MS-DOS and OS/2 programming.

Lattice
Subsidiary of SAS Institute Inc.

Lattice, Incorporated
2500 S. Highland Avenue
Lombard, IL 60148
Phone: 800/533-3577
In Illinois: 312/916-1600

Lattice is a registered trademark of Lattice, Incorporated. MS-DOS is a registered trademark of Microsoft Corp. OS/2 is a registered trademark of International Business Machines Corp.

CIRCLE NO. 209 ON READER SERVICE CARD

THE BEST OF BOTH WORLDS

Developing an application used to be easy — all you had to do was program it. But today, with countless languages, compilers, libraries, databases, editors, debuggers, and other tools, it is choosing the right development software that creates the real problem.

Now *The Andsor Collection* introduces a unique solution: a collection of sophisticated development tools, which you can use on their own, or together with your old ones.

The Andsor Collection is, of course, the superb application development system that programmers, VARs, and other developers have been using for over two years. And starting with Version 2.2, *The Andsor Collection* has acquired a new dimension: now you can access all its functions from within another program!

Think of it as a comprehensive, universal, language independent library. But *The Andsor Collection* is not a collection of subroutines: it is a seamless, integrated, interactive environment, specifically designed to expedite application development.

Whether you use C, Pascal, Cobol, Fortran, Basic, or any other language, *The Andsor Collection* can enhance your applications dramatically. Whether you add functions to an old program, or you write a new one, you can make them faster, more efficient, and more appealing.

Use *The Andsor Collection* to implement an entire application, or just portions of an application. You can, for example, create a windowed environment, add attractive data entry functions, define indexed data file structures, produce sophisticated reports or forms, and so on.

System Features

The Andsor Collection is the most versatile application development environment. And when using it with your programs, all its countless features can become part of your application. Hundreds of commands, functions, and options, are available to help you implement any application.

No list can be complete, so here are just some of these features: powerful database functions, maintenance-free multi-index data files, variable length data fields, unlimited file relations, complete window management, unique text processing functions, flexible data tables, powerful inquiry and reporting functions, versatile data entry capabilities, flexible procedural language, automatic error handling, extensive computational capabilities, data analysis and statistics, unique programmable charts, many printing functions, data communications, convenient system log file, full control over color attributes. And much, much more.

How It Works

This is simple and ingenious. Your program and *The Andsor Collection* reside in memory together (you load *The Andsor Collection*, which then loads your program). To transfer control to those portions of the application implemented in *The Andsor Collection*, you simply issue a software interrupt in your program, exactly the way DOS and BIOS functions are called. (If you are not familiar with this, examples in the manual show you how to do it.)

While in *The Andsor Collection*, the operation is identical to its operation when used alone. Finally, one command returns control to your program. And if you need this, simple commands transfer data directly to and from memory areas in your program (a number of formats are possible, all in standard ASCII, compatible with any language). Both *The Andsor Collection* and your program run as ordinary DOS programs. And you can also use them with other software (such as permanently resident programs).

Although *The Andsor Collection* has far more features than other development systems, it is only one tenth their size. So the entire system can stay in memory, keeping all functions instantly accessible.

And *The Andsor Collection* is famous for its unique interactive environment. There is no conversion or translation — modify a procedure, a file definition or relation, a data entry screen, or anything else, and the change takes effect immediately, even while the application is running! Application development is a new experience.

The application users will benefit too. *The Andsor Collection* is amazingly fast, and since all data is in variable length format, the files take a fraction of the space needed with other systems. So not only will you develop your applications sooner, but they will be more efficient too. Whether you use *The Andsor Collection* alone, or to enhance a program.

So get the best of both worlds. Order *The Andsor Collection* today, and discover a whole new environment, without giving up your old development tools or your existing applications. Moreover, *The Andsor Collection* will be useful with all your future applications and languages.

You won't find a better value in development software: one program that is both a powerful stand-alone application development system, and a unique language independent collection of software tools; plus the run-time interpreter with unlimited royalty-free distribution. All for an incredibly low price. And with our 60 day money back guarantee*, you have little to lose and a lot to gain.

"We looked at several systems and decided to standardize on The Andsor Collection. We are using it in many applications in several departments."

Gordon Eyers, Director of Information Services, Public Utilities Commission, Brantford, Ontario

"With The Andsor Collection we have achieved faster development and more efficient applications, which is important in large and complex projects like our Court Management System."

Dr. Mark Schrager, Consultant, Municipal Computer Services, Rochester, New York

"The Andsor Collection is unequalled when we need a solution in a hurry. Applications that we have implemented include modeling, data acquisition and analysis, and reporting."

Joe Blask, Engineering Support, General Motors, Troy, Michigan

ANDSOR® — *The Andsor Collection™*

ANDSOR RESEARCH INC.
390 Bay Street, Suite 2000
Toronto, Ontario M5H 2Y2
(416) 245-8073

To order call toll free
(U.S. and Canada)

1-800-628-2828
Ext. 535

$145 *(includes shipping*)*

Visa, MC, AmEx, Check

*Price includes shipping in the U.S. and Canada. Please add $10 for shipping to other countries. If you return the software, $8 will be deducted from the refund, to cover our shipping cost.

System requirements: any IBM PC or PS/2 or fully compatible, 250K+ (excluding DOS and other programs), one disk drive or hard disk, monochrome or color monitor, DOS 2.0+ or OS/2

CIRCLE NO. 210 ON READER SERVICE CARD

(continued from page 138)

ments. Still, it saves time when you don't have to drop out of the editor to compile, then reenter it to fix errors.

I use Brief a lot, and aside from the window-changing problem mentioned earlier, there's only one thing about it that really annoys me. That is that you can't use the gray asterisk to type the multiplication symbol; you must use Shift-8. Brief regards the gray asterisk as a synonym for ALT-U (Undo). I should probably reconfigure the keyboard, which Brief lets you do, but I haven't gotten around to it.

Brief comes on two floppies and with two spiral-bound manuals in a slipcase: a 161-page editor reference and a 195-page macro language guide. Both contain comprehensive tutorials written in English. The 36-page editor tutorial gets you productive with this excellent programmer's tool in about an hour.

Like I said, it's terrific.

by Kent Porter

Guidelines C++

Product:
Guidelines C++, Version 1.2
Target:
PC and compatibles
Requires:
DOS 2.1 or later; 800K of hard disk space; 640K of RAM; and Microsoft C Version 3.0 or later
Pricing:
$295;
owners of previous versions $25
Vendor:
Guidelines, P.O. Box 749, DDJR
Orinda, CA 94563
(415) 254-9393 or (415) 254-9183

If you've been programming in C for a while, enjoyed structures and pointers to functions, and realized that there is something someone could do to structure the programming even more, you've likely already postulated a language like C++. C++ is still fairly new (sort of like C was in 1987), not terribly popular (yet), and extremely powerful.

C++ is an object-oriented language, which means, in short, that more emphasis is placed on the design of objects and object manipulators than on functions and procedures. Like C, C++ was deveoped by AT&T. Version 1.2 is their third release following Version 1.0 and 1.1. Guidelines' C++ for MS-DOS Version 1.2 is ported from the AT&T release of the same number.

It's important to note that the C++ translator is not a compiler, but rather works in conjunction with one. A C++ program is first preprocessed, then translated to C, then compiled by a C compiler. C++ is, though not strictly, a superset of C and most C programs will flow through the preprocessor and translator unchanged. In fact, some of the extensions to C that the ANSI committee has adopted were introduced in C++.

I found this package easy to install and flexible enough to deal with such things as alternate drives and directories than those recommended. I would have liked the option to install the package in the same directories as the C compiler it worked with, but it's easy to rearrange the directories later.

Several subdirectories are created within the main program directory. The LIB directory holds the libraries, one each for the five standard memory models. The INCLUDE directory contains all of the AT&T supplied headers, modified only enough to avoid conflict with the Microsoft headers. The SZAL directory holds information for each memory model about the size and alignment of different sized objects, none of which you should ever need worry about. The SCR directory contains a directory for each of the chapters in the Stroustrup book, plus the source code for a batch file builder. The DOC directory contains documentation on the C++ preprocessor and the C++ translator, as well as a copy of the tutorial chapter of the manual.

The last directory created is the BIN directory which contains the .EXE files for the C++ preprocessor and translator, as well as the batch file builder. Also in the BIN are 24 batch files (built with batch file builder) that take .CPP files to .CI (preprocessing done), to .CC (translator output), to .OBJ (compilation complete), or to .EXE for each of the five memory models plus a default.

Batch files are provided to allow the translator more working memory. Of course, MAKE could be used to invoke all of the appropriate programs individually, but more memory is absorbed by the MAKE utility than by the built-in batch file processor.

The Bjarne Stroustrup book, *The C++ Programming Language*, is included but may be omitted to save $10. The AT&T release notes are also provided which document differences between C and C++ as well as the differences between 1.0, 1.1, and 1.2 of C++. An installation guide and tutorial (really a series of examples) round out the documentation.

Free support is by mail only, unless you pay $50 per year to have access to telephone technical support. I'd prefer that they allowed some sort of phone access for a short period after acquisition (NOT registration). On the other hand, $50 doesn't seem unreasonable, and in the worst case, you can think of the price of the package as $345 and purchase the first year at the same time as the package.

Finally, from my contact with them, Guidelines seemed like reasonable people and I suspect that they would allow some simple questions to be answered without charge. At least they answered the questions I had during installation. I found few glitches in the package, none serious, just annoying. I liked the fact that all of the chapter directories included a makefile, however, the makefile .SUFFIXES command was not quite complete lacking the .EXE extension. This "bug" has been reported and will likely be fixed long before you get yours. Overall, I liked the package, and given sufficient free time to learn yet another language, I'll use it regularly.

by Richard Relph

(continued from page 14)

For example, I prepared the minimal dynalink module shown in Example One, and compiled it with IBM C/2 using the options *–c –Gs –-ALu* which, literally translated, mean *compile only, no stack checks, large model with no dependency on DS*. Then I tried to link the object module with the definition file

```
LIBRARY APROC
EXPORTS aproc
```

which requests linking a dynamic link library. The link reported an unresolved external of *__acrtused*, which I recognized as an entry point into the C setup routines. When I supplied the C runtime library, the link pulled in all of the C setup code and generated an unresolved extern for *__main!* I circumvented this by adding the line

```
void __acrtused( ){ }
```

to my module. This satisfied the external reference, and in fact the generated code did not actually call the dummy *__acrtused* function.

Thus finagled, the module did link and would have worked as a dynamic link package. But when I changed it to use a floating point number, as shown in Example Two, the link failed with many unresolved externs in the float libraries. These float routines are known not to be reentrant, so they can't be used in dynamic link code. Neither can the C library routines for file I/O.

In short, it is possible to build dynamic link packages using the official OS/2 C compilers. You have to know the ins and outs of the compiler very well, restrict the code to integer arithmetic, and do all I/O by way of calls on OS/2 system functions instead of the C library (this last is no great hardship, as the OS/2 system functions are quite complete). With other languages and compilers, good luck!

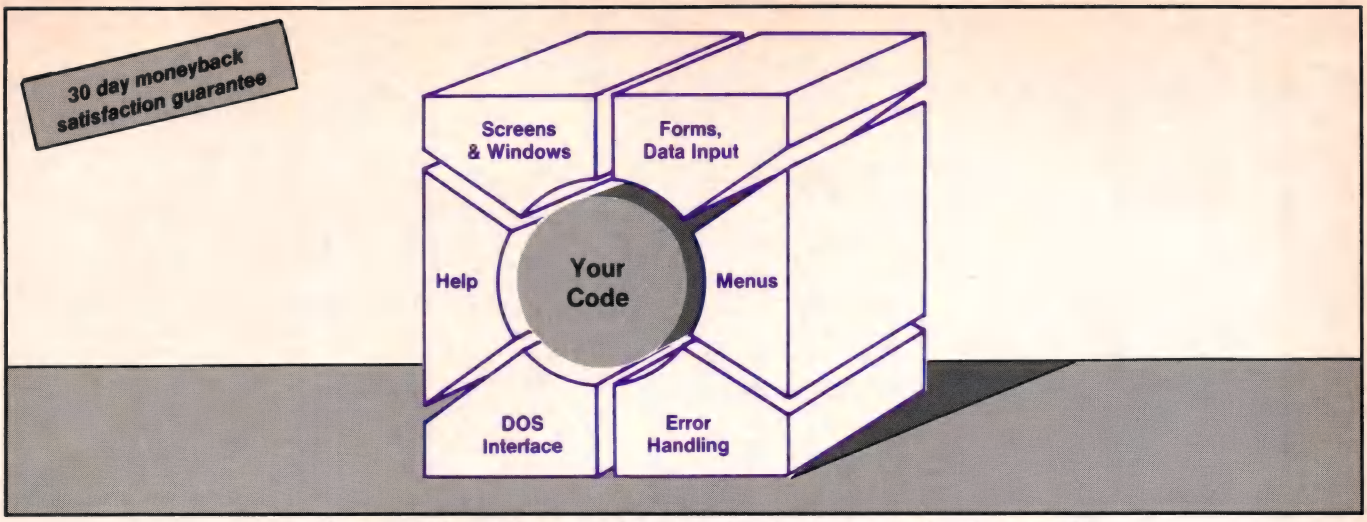
DDJ

```
int far pascal aproc(char far *arg1)
{
char locv[8];
        locv[2] = *arg1;
        return(locv[2]);
}
```

Example 1

```
void _acrtused(){}
static double bignum = 0.0;
int far pascal aproc(char far *arg1)
{
        bignum += *arg1;
        return(bignum);
}
```

Example 2



C-Worthy Interface Library helps you smoothly pull together all aspects of an excellent Human Interface.

C Programmers: Wrap an Exciting, Bullet-Proof Interface Around Your Code Quickly.

Introducing... C-Worthy® Interface Library

The only human interface package you need. That's what our customers are telling us. One early adopter, Novell, Inc. uses it exclusively in the development of their NetWare® Utilities, which reach over 500,000 users. You see, C-Worthy Interface Library is the only library available to handle every aspect of your program's human interface, all in one package. Now your programs will have a consistent look and feel. You no longer have to integrate pieces of libraries from different manufacturers.

As important as you know users are, you often don't have the time to heavily invest in writing routine code. And that's OK, because with over 400 tight, ready-to-use functions, C-Worthy Interface Library takes care of the tedium and lets you spend your time doing what you enjoy. Concentrate on the heart of your application — features that make it unique, special. Let C-Worthy Interface Library do your:

- Menus
- Error Handling
- DOS Interface
- Context Sensitive Help
- Screens, Windows
- Forms, Data Input (optional)

You control color, size, border, location, etc. And if there's anything you want to change, you can. Source is available to provide you with the flexibility you need. And you can distribute your applications freely, with no royalties.

C-Worthy Interface Library requires hard disk media with 256K RAM. MSDOS 2.0 + and IBM PC, or compatible, TI Professional, NEC APC III, or VICTOR 9000. C-Worthy is a registered trademark of Custom Design Systems, Inc.

Tech Specs

- Compilers: Microsoft 3.0+, Quick, Turbo, Lattice. All models.
- 350+ functions written in C, 75+ in Assembler.
- Menus: Fully support pop-up, Lotus style, MS Windows style (pull-down), pull-up.
- Errors: DOS, program, and user.
- DOS Interface: 62 functions. File handling, dir. and drive management, date & time conversion, wildcards, more.
- Help: System and context sensitive.
- Screens: Screen display, color palettes, save, restore, scroll, more.
- Windows: Exploding, tiled, pop-up, overlapping. Direct video access and virtual. Up to 50 active at any time.
- Keyboard Handling: Regular, function, interrupt, background procedures.
- Editing: String and word wrap text.
- Form Interface Library: 118 functions. Over 15 field types, and user definable field types. 3 levels of data validation: type, multiple field ranges, optional validation procedures. Hide, lock, or secure a field. Optimal field movement.
- Foreign Languages: All text messages in separate files for easy translation.
- Compatible with MS Windows.
- OS/2 special overlay when released.
- Machines: Autodetect for MDA, CGA, EGA, VGA, TI, AT&T, Victor.
- No royalties.

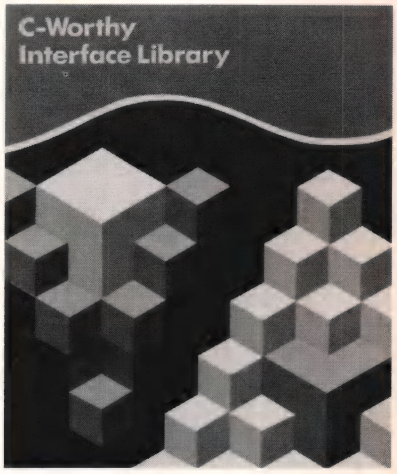
"I heartily recommend this package."
— David A. Schmitt, president, Lattice, Inc.
Over 400 developers in 16 countries already use it.

Thorough Documentation

Indexed alphabetically and by category, the 700+ page Reference Guide includes for each function: an example, description, calling conventions, return values, and related functions. The 250 page User's Guide gets you going with its tutorial and "Getting Started" sections.

CIRCLE NO. 213 ON READER SERVICE CARD

"C-Worthy is a comprehensive C library whose time has come. I heartily recommend it as your next purchase." —Computer Language, 8/87



C-Worthy Interface Library:

Object only	**$ 195**
Form Interface Library add-on	**$ 100**
Object with Forms	**$ 295**
Object with Forms & Library Source	**$ 495**

Please specify compiler and version when ordering.

To Order Call

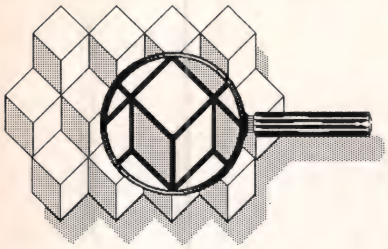
(800) 821- 2492

in MA (617) 337-6963

Solution Systems™

541-D Main Street, Suite 410
South Weymouth, MA 02190

OF INTEREST



Languages

Coral Software Corp. and **Franz Inc.**, in a combined development effort, have announced Allegro CL, a complete implementation of Common LISP for the Mac II. The product features an incremental native code compiler, object-oriented programming capabilities, a programmable EMACS-style editor, and advanced debugging tools such as a window-based inspector and stepper. These are fully integrated into the Macintosh user interface to create an unparalleled LISP programming environment. Thus, Allegro CL allows the Mac to function as an independent AI workstation.

Allegro CL requires 1-Mbyte RAM and 1.6 Mbyte of disk storage. It runs on both 68000 and 68020 CPUs, directly supports the 68881 math coprocessor, and runs on Macintoshes with large screens. The product sells for $399.95. Reader Service No. 24.
Coral Software
P.O. Box 307
Cambridge, MA 02142
(617) 547-2662

The Golden Common LISP (GCLISP) Developer 286, Version 3.0 is available from **Gold Hill Computers**. The new version features a compiler which strictly adheres to the complete semantics of Common LISP, faster load times, an expanded GMACS editor for easier manipulation of LISP syntax, and support for Portable Common Loops, a flexible object programming system. The GCLISP Developer also comes with the San Marco LISP Explorer, an interactive tutorial developed by San Marco Assoc.; the GCLISP Reference Manual; User's Guide; the second edition of LISP by Winston and Horn; and the Common LISP Reference Manual by Guy Steele. Reader Service No. 25.
Gold Hill Computers
163 Harvard St.
Cambridge, MA 02139
(617) 492-2071

PowerLisp Version 2.0, from **MicroProducts,** offers complete support for Common LISP and InterLISP-10 as well as Common LOOPS. PowerLisp supports the complete Common LISP standard. The product is available for the IBM PC AT, Intel 386, and VAXmate. PowerLisp for the 286 is $2,195 and for the 386 is $2,995. Reader Service No. 26.
MicroProducts
370 W. Camino Gardens Blvd.
Boca Raton, FL 33432
(800) 553-0777

Digitalk is now shipping release 2.0 of Smalltalk/V, a PC-based implementation of the Smalltalk programming language. The new release supports the high-resolution graphic modes (640 × 480) of the IBM PS-2/25 and 30 computers and includes windowing enhancements.

Smalltalk/V Release 2.0 is priced at $99.95. Registered owners of earlier releases may obtain upgrades for $25. Reader Service No. 27.
Digitalk
9841 Airport Blvd.
Los Angeles, CA 90045
(213) 645-1082

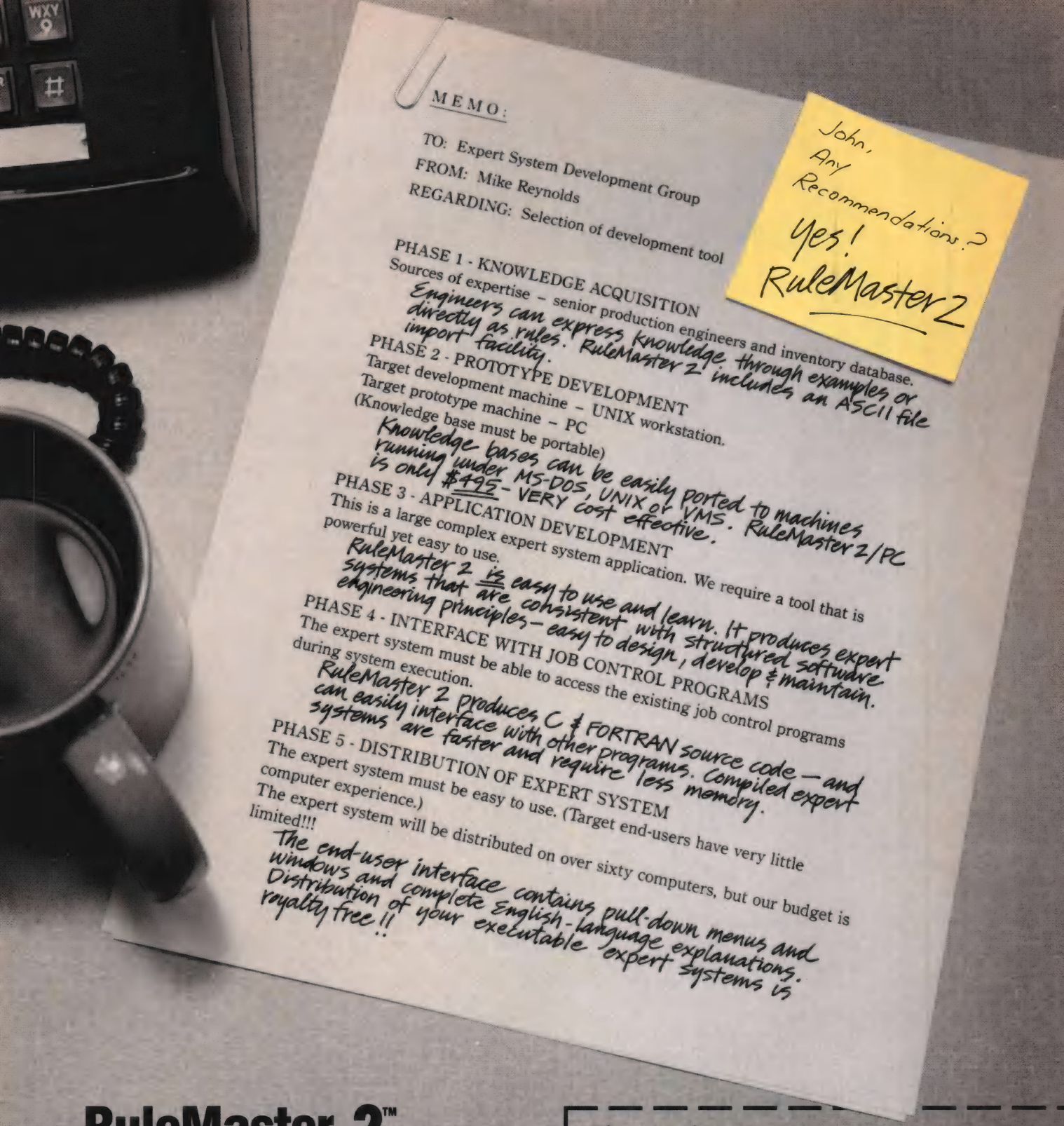
Trilogy is a new programming language developed by **Complete Logic Systems** that combines procedural programming, declarative programming, and database programming. Trilogy comes with its own environment including editor, module library, interactive compiler (producing native 8086 and 8087 code), online linker, loader, and context sensitive help screens. Users can edit, compile, link, and execute multimodule programs without ever leaving the Trilogy environment. The product also comes with four standard modules called Math, Strings, Files, and Windows. These modules export routines (in that order) for transcendental functions, string/date/time manipulation functions, file access functions, and windowing functions.

Trilogy runs on the IBM PC, IBM PC AT and XT, and compatibles using DOS 2.1 or later, uses 512K, and is not copy protected. It is priced at $99.95 US. Reader Service No. 28.
Complete Logic Systems
741 Blueridge Ave.
North Vancouver, B.C.
Canada V7R 2J5
(604) 986-3234

Books

McGraw-Hill has published a new book called *A Comprehensive Guide to AI and Expert Systems: Turbo Pascal Edition* by Robert Levine, Diane E. Drang, and Barry Edelson. The book offers full explanations of the basic concepts of artificial intelligence and expert systems and also shows how AI techniques can be implemented on a personal computer. Complete discussions examine a wide range of important topics, including natural language processing, forward and backward chaining, and the use of probability and fuzzy logic in expert systems. Aspects such as object-oriented expert systems, semantic nets, certainty factors, automated learning, using PROLOG to design expert systems, and LISP are also explored. The book sells for $19.95. Reader Service No. 29.
McGraw-Hill Book Company
11 W. 19th St.
New York, NY 10011
(212) 337-5945

Structured Induction in Expert Systems, by Alen D. Shapiro, is a new book published by **Addison-Wesley** which shows how the knowledge acquisition process can be automated by the application of inductive learning techniques. It provides a guide to techniques for the synthesis of transparent rules from examples supplied by the domain expert. It also shows applications of these techniques to complex problems in



RuleMaster 2™

the right tool for all phases of expert system development

RADIAN CORPORATION
(512)454-4797

A company of The Hartford Steam Boiler Inspection and Insurance Company

RuleMaster 2 is a trademark and RuleMaster is a registered trademark of Radian Corporation.
MS-DOS is a registered trademark of Microsoft Corporation. UNIX is a registered trademark of AT&T Bell Laboratories, and VMS is a registered trademark of Digital Equipment Corporation.

CIRCLE NO. 214 ON READER SERVICE CARD

Please send me:

☐ additional information on RuleMaster 2
☐ a RuleMaster 2 demonstration disk ($10.00) ☐ 5¼" ☐ 3½"

Payment: check☐ VISA☐ Mastercard☐

Card No. _____ Exp. Date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Phone _____

Send to: RuleMaster® / #270 / P.O. Box 9802 / Austin, TX 78766-0802

OF INTEREST
(continued from page 144)

chess end-game classification. This book can be used as a supplement for graduate-level courses on expert systems and AI programming. The suggested retail price for the book is $31.25. Reader Service No. 30.
Addison-Wesley Publishing
Reading, MA 01867
(617) 944-3700

Expert Systems

OPS83, by **Production Systems Technologies** is a rule-based programming language that was designed as a tool for developing and delivering knowledge-based, second generation expert systems. The language is not restricted to merely diagnostic or classification tasks. Code written in OPS83 can easily be interfaced to code written in other languages. Reader Service No. 31.
Production Systems Technologies
642 Gettysburg St.
Pittsburgh, PA 15206
(412) 362-3117

VP-Expert from **Paperback Software** is a rule-based expert system development tool. Features include a built-in text editor, an inductive front end that creates if-then rules directly from examples in external files of other programs, an inference engine, floating-point arithmetic and trigonometric functions, confidence factors, text and graphic tracing, and external program calls.

VP-Expert runs on the IBM PC, IBM PC AT and XT, IBM PS/2, or compatibles with 384K, uses MS-DOS 2.0 or later, and is not copy protected. The product sells for $124.95. Reader Service No. 32.
Paperback Software International
2830 Ninth St.
Berkeley, CA 94710
(415) 644-2116

The Berkshire Software Company has released Turbo Shell Version 2.0, a software system designed to provide a complete environment for the development of expert systems. The product provides menu-driven facilities for creating, modifying, and consulting knowledge bases comprised of rules, facts, and solutions. The system is based on the

Apollo, H-P, and Sun Users; and now, for the MAC II, too.

AdaNow

New Alsys Toolset For 68000 Ada Builds Unique Project Environment

Organizations serious about the 680X0 architecture, and serious about working with the government, want a lot more than just validated Ada compilers. They want quality solutions; production quality compilers and quality programming tools.

Just what Alsys offers. Alsys' new 68000 Ada Developer's Toolset includes:

- **AdaProbe**, a unique source-level symbolic debugger and program viewer;
- **AdaXref**, an inter-unit cross-referencing utility;
- **AdaReformat**, a pretty printing tool for reformatting source files to selectable conventions; and
- **AdaMake**, an automatic recompilation facility.

Consider, too, all those special Ada "manager tools" that are part of the Alsys Version 3 compilation system: the **Family Manager**, the **Unit Manager**, and the **Library Manager**.

Together, they implement the new Alsys **Multi-Library Environment** that allows teams of programmers to share thousands of logically organized compilation units.

Alsys 68000 compilers are in a class by themselves; highest code quality, maturity, reliability, robustness, superior optimization technology, unexcelled error messages...And now, with the new development tools, they are at the core of an Ada project environment unique in the industry.

Alsys 68000 compilers and our new 68000 Ada Developer's Toolset are now available for the Apollo Domain, Sun 3, Apple Macintosh II, and H-P 9000/ Series 300.

Ada is NOW. Alsys solutions are NOW. Call or Write.

alsys

In the US: Alsys Inc., 1432 Main St., Waltham, MA 02154 Tel: (617) 890-0030
In the UK: Alsys Ltd., Partridge House, Newtown Rd., Henley-on-Thames, Oxon RG9 1EN Tel: 44 (491) 579090
In the rest of the world: Alsys SA, 29 Avenue de Versailles, 78170 La Celle St. Cloud, France Tel: 33 (1) 3918.12.44

The Many Facets of **Quality**

alsys

_____ YES, Send more information on the Toolset and your 68000 compilers.

_____ Send me your free brochure, *The Many Facets of Quality.*

Name_____

Company_____

Address_____

City_____ State_____ Zip_____

Phone_____

Alsys, Inc. • 1432 Main Street • Waltham, MA 02154

DDJ 4/88

CIRCLE NO. 216 ON READER SERVICE CARD

*Ada is a registered trademark of the U.S. Government (AJPO).

classic MYCIN model of expert system development using production rules for the classification of knowledge. Turbo Shell can be requested to provide an explanation of its logic and line of reasoning in pursuit of a conclusion, and will provide a complete explanation when a conclusion has been reached.

The Turbo Shell expert system development environment is written entirely in Turbo PROLOG. The package sells for $89 and runs on IBM PCs and compatibles with one dual sided floppy-disk drive and at least 256K of RAM. Reader Service No. 33.
The Berkshire Software Company
44 Madison St.
Lynbrook, NY 11563
(516) 593-8019

Miscellaneous

AI-NET 101 from **AI Ware** is a data analyzer based on neural net technology. It autonomously learns patterns among input samples using techniques inspired by knowledge of the human brain's learning processes. AI-NET 101 operates in real-time and is suitable for integration into industrial environments. Its modular design allows several nets to be connected into a single system. AI-NET 101 uses proprietary software that takes advantage of the neural net's internal structure to increase its learning performance. AI-NET 101 uses a Microsoft Windows interface. The AI-NET accelerator card, which fits into the IBM PC, increases learning speeds up to 10 times and more. The product sells for $4,500. Reader Service No. 34.
AI Ware Inc.
11000 Cedar Ave., Ste. 212
Cleveland, OH 44106
(216) 421-2380

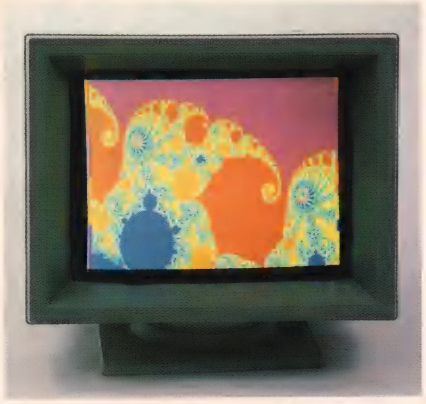
CLOE is a new software package developed by **Symbolics** that is a software environment for both the Symbolics 3600 product line and Intel 80386-based personal computers. CLOE consists of three software modules—one for delivery and two for application development and fine tuning. For delivery, CLOE Runtime executes the application on the targeted 80386-based delivery machine. This software module is a native 80386 Common LISP runtime platform with extensions such as multitasking, support for new flavors object-oriented programming language, advanced exception handling facilities, and a high-performance garbage collector. For development, programmers will need the 3600-based Developer software module and the companion 80386-based Application Generator.

CLOE Runtime requires an Intel 80386-based PC or workstation, Unix System V, Release 3.0, 2 Mbyte of RAM and a 20-Mbyte hard disk. Prices vary according to licensing configuration. Reader Service No. 35.
Symbolics Inc.
11 Cambridge Center
Cambridge, MA 02142
(617) 621-3727

DDJ

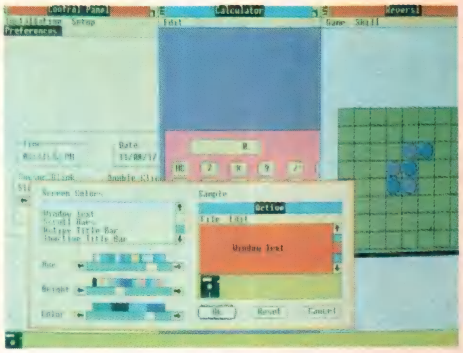
VERY HIGH RESOLUTION

The PC Tech COLOR and MONOCHROME video processor boards employ the TMS 34010 high performance graphics co-processor to insure the best possible video performance at reasonable prices.



Color 34010 Video Processor:

- **Featured on the cover of Micro Cornucopia.**
- **From 800 x 512 through 1024 x 800 resolution (depending on monitor and configuration).**
- **8 Bits per pixel for 256 simultaneous colors**
- **Hardware support for CGA/MDA emulation.**
- **PC, XT, and AT compatible**



The PC Tech Color 34010 video processor is a superior 34010 native code and DGIS development tool. We support up to 4 megabytes of program (non-display) 34010 RAM as well as up to 768K bytes of display RAM. ***Compare our architecture and prices to any other intelligent graphics board. Then choose the PC Tech Color 34010 Video Processor for your development engine and your production requirements as well.***

Color 34010 Video Processor $1,195.00

Price includes 512K display RAM, 1024K program RAM, and utility software. Monitor not included.
Also available: DGIS, 34010 C compiler, assembler, 34010 fractal software,
additional display and program memory, and various monitor options.

PC Tech Monochrome 34010 Video Processor and Monitor

- **736 x 1024 resolution (other options available)**
- **2 bits per pixel for 4 hardware gray shades**
- **Hardware support for CGA/MDA/Hercules emulation**
- **PC, XT, and AT compatible**
- **Full page 66 line text editing with many popular editors**
- **Excellent windows 2.0 application development system**

The graphics and bit manipulation capabilities of the TMS 34010 make the PC Tech Monochrome 34010 Video Processor 66 line full page text and graphics display faster than many 25 line systems. The video processor is available separately or with the high resolution white phosphor monitor shown above.

Monochrome 34010 Video Sub-System $1,295.00

Price includes Monochrome Video Processor and monitor pictured above.
Also available: DGIS, TI 34010 C compiler, TI assembler.
Monochrome 34010 Video Processor also available separately.

Designed, Sold and Serviced By:

PTech

904 N. 6th St.
Lake City, MN 55041
(612) 345-4555
(612) 345-5514 (FAX)

CIRCLE NO. 218 ON READER SERVICE CARD

Upgrade Your Technology

We're Programmer's Connection, the leading independent dealer of quality programmer's development tools for IBM personal computers and compatibles. We can help you upgrade your programming technology with some of the best software tools available.

Comprehensive Buyer's Guide. The CONNECTION, our new Buyers Guide, contains prices and up-to-date descriptions of *over 600 programmer's development tools by over 200 manufacturers*. Each description covers major product features as well as special requirements, version numbers, diskette sizes, and guarantees.

How to Get Your FREE Copy: 1) Use the reader service card provided by this journal; 2) Mail us a card or letter with your name and address; or 3) Call one of our convenient toll free telephone numbers.

If you haven't yet received your copy of the Programmer's Connection Buyer's Guide, act now. Upgrading your programming technology could be one of the wisest and most profitable decisions you'll ever make.

USA 800-336-1166

Canada 800-225-1166
Ohio & Alaska (Collect) 216-494-3781
International 216-494-3781
TELEX 9102406879
FAX 216-494-5260

Business Hours: 8:30 AM to 8:00 PM EST Monday through Friday
Prices, Terms and Conditions are subject to change.

PROGRAMMER'S CONNECTION — Service — QUALITY INTEGRITY SUPPORT PRICE

ORDERING INFORMATION

FREE SHIPPING. Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Call for express shipping rates.

NO CREDIT CARD CHARGE. VISA, MasterCard and Discover Card are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include expiration date and authorized signature.

NO COD OR PO FEE. CODs and Purchase Orders are accepted at no extra cost. No personal checks are accepted on COD orders. POs with net 30-day terms (with initial minimum order of $100) are available to qualified US accounts only.

NO SALES TAX. Orders outside of Ohio are not charged sales tax. Ohio customers please add 5% Ohio tax or provide proof of tax-exemption.

30-DAY GUARANTEE. Most of our products come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

SOUND ADVICE. Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

INTERNATIONAL ORDERS. Shipping charges for International and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. International orders (except Canada), please include an additional $10 for export preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

MAIL ORDERS. Please include your telephone number on all mail orders. Be sure to specify computer, operating system, diskette size, and any applicable compiler or hardware interface(s). Send mail orders to:

Programmer's Connection
Order Processing Department
7249 Whipple Ave NW
North Canton, OH 44720

386 products

	List	Ours
386 ASM/386 LINK *Cross Asm by Phar Lap*	495	377
386 DEBUG *Cross Debugger by Phar Lap*	195	129
NDP C-386 *by MicroWay* ..... *New*	595	529
NDP ForTran-386 *by MicroWay* ..... *New*	595	529
PC-MOS/386 *Single-User by The Software Link*	195	155
PC-MOS/386 *5-Users by The Software Link*	595	539
PC-MOS/386 *25-Users by The Software Link*	995	869

alsys products

Ada *GSA-validated w/maintenance*	3355	2975
Ada Developer's Toolset *Volumes 1 & 2*	995	919
AdaQUERY	200	185

american software int'l

DMS Resident-ASM	150	139
DMS Resident-C	150	139
DMS Screen Master ..... *New*	200	185

1st-CLASS by Programs in Motion

List $495 Ours $399

1st-CLASS is a knowledge analysis tool that allows you to see what data is relevant in your decision making process. Data can be entered in a familiar spreadsheet format or directly as a decision tree. The graphic decision tree that you build allows you to clearly view your logic every step of the way. With 1st-CLASS, you can exchange data with databases and spreadsheets, interface it to other programs, operate hardware, read instrumentation, or use 1st-CLASS as a logic engine within your own program.

assembly language

Cross Assemblers *Various by 2500 AD*	CALL	CALL
OPTASM *by SLR Systems*	195	179

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS PLUS/5.0	129	99
KeyPlayer *Super Batch Program*	50	45
LIGHT TOOLS *for Datalight C* ..... *Sale*	100	55
PASCAL TOOLS/TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
Turbo ASYNCH PLUS/4.0	129	99
Turbo C TOOLS	129	99
Turbo POWER TOOLS PLUS/4.0	129	99
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

EUREKA *Equation Solver*	167	105
Paradox 1.1 *by Ansa/Borland*	495	359
Paradox 2.0 *by Ansa/Borland*	725	525
Paradox Network Pack *by Ansa/Borland*	995	725
Quattro: The Professional Spreadsheet	195	125
Reflex: The Analyst	150	99
Sidekick	85	57
Superkey	100	64
Turbo Basic Compiler	100	64
Turbo Basic Database Toolbox	100	64
Turbo Basic Editor Toolbox	100	64
Turbo Basic Telecom Toolbox	100	64
Turbo C Compiler	100	64
Turbo Lightning	100	64
Turbo Lightning Word Wizard	70	47
Turbo Pascal	100	64
Turbo Pascal Database Toolbox	100	64
Turbo Pascal Developer's Toolkit	395	259
Turbo Pascal Editor Toolbox	100	64
Turbo Pascal Gameworks Toolbox	100	64
Turbo Pascal Graphix Toolbox	100	64
Turbo Pascal Numerical Methods Toolbox	100	64
Turbo Pascal Tutor	70	41
Turbo Prolog Compiler	100	64
Turbo Prolog Toolbox	100	64

c language

Eco-C88 Modeling Compiler *by Ecosoft*	100	69
Lattice C Compiler *vers. 3.2 from Lattice*	500	265
Optimum-C *by Datalight*	139	95

Peabody Pop-Up Reference Utility by Copia International

List $100 Ours $89

Peabody is a fast and flexible on-line reference utility with databases available for Turbo Pascal , Turbo C, Microsoft C, or MS DOS. It provides instant, accurate and complete language information in pop-up frames at the touch of a key. With Peabody, you can select general topics from a structured subject menu, or use Peabody's hyperkey to get instant help for the keyword closest to the cursor. Specify database desired. Additional databases are available for $100 with manual or $50 without manual.

c utilities

BTree *by Softfocus* ..... *New*	75	69
ISAM *File Manager* ..... *New*	40	37
C++ *by Guidelines*	295	259
c-scape *by Oakland Group* ..... *New*	299	279
C_talk *by CNS* ..... *New*	150	119
CBTREE *by Peacock Systems* ..... *New*	159	98
CQL *by Machine Independent Software* ..... *New*	395	329
Curses Window Dev Pkg *by Aspen Scientific*	119	105
with Source Code	289	249
dBx *dBASE to C Translator by Desktop AI*	CALL	CALL
with Source Code	CALL	CALL
Entelekon Combo Package ..... *Clearance*	200	99
FOR_C *by Cobalt Blue* ..... *New, Special Price*	650	559
GraphiC *by Scientific Endeavors*	395	309
Interwork *by Block Island Tech* ..... *New*	129	115
PANEL *by Roundhill, Specify QuickC or Turbo C*	129	95
PANEL Plus *by Roundhill*	495	395
PC/Forms *by Golden Software* ..... *New*	150	129
QPARSER+ *by QCAD Systems* ..... *New*	475	CALL
RTC PLUS *Fortran to C by Cobalt Blue*	450	369
Vitamin C *by Creative Programming*	225	149
VC Screen *Forms Designer*	100	79
WKS LIBRARY *by Tenon Software*	129	109

database management

Clipper *by Nantucket*	695	379
dBASE III Plus *by Ashton-Tate*	695	389
dFLOW *by Wallsoft*	149	119
FoxBASE+ *by Fox Software*	395	249
FoxBASE+/386 *by Fox Software* ..... *New*	595	399
Genifer *by Bytel*	395	249
MAGIC PC *by AKER*	199	167
Q&A *by Symantec*	349	219
R:Base 5000 *by Microrim*	495	359
R:Base System V *by Microrim*	700	439
UI Programmer *by Wallsoft*	295	239

debuggers

Periscope I *with Board by Periscope*	345	275
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *8 MHz version*	995	795
Periscope III *10 MHz version*	1095	875

digitalk products

Smalltalk/V	100	84
EGA/VGA Color Option	50	45
Goodies Diskette #1	50	45
Goodies Diskette #2 ..... *New*	50	45
Goodies Diskette #3 ..... *New*	50	45
Smalltalk/Comm	50	45
Smalltalk/V 286 ..... *New*	200	159

dos utilities

Desqview *from Quarterdeck*	130	105
Mace Utilities *Paul Mace Software*	99	85
XO-SHELL *by Wyte Corporation*	49	45

elan computer products

Eroff ..... *New*	695	589
NROFF/PC ..... *New*	99	89

essential products

Breakout Debugger	125	89
C Utility Library	185	118
Essential Communications	185	118

CIRCLE NO. 219 ON READER SERVICE CARD

Essential Communications *with Break Out*	250	**189**
Essential Graphics	250	**183**
/*resident_C*/	99	**85**
with Source Code	198	**148**
ScreenStar	99	**85**
with Library Source Code	198	**154**

faircom products

c-tree & r-tree Combo	650	**519**
c-tree *ISAM File Manager*	395	**315**
r-tree *Report Generator*	295	**239**
d-tree	CALL	**CALL**

gimpel products

C-terp *Specify compiler*	298	**219**
for UNIX/XENIX	498	**379**
PC Lint	139	**99**
Turbo C-terp *for Turbo C*	139	**119**

golden bow products

Vcache	50	**47**
Vfeature *Hard Disk Utility*	80	**74**
Vfeature Deluxe	120	**111**
Vopt *Hard Disk Optimization Utility*	50	**47**

greenleaf products

Greenleaf C Sampler *specify QuickC or Turbo C*	95	**69**
Greenleaf Comm Library	185	**125**
Greenleaf Data Windows Library	225	**155**
with Source Code	395	**249**
Greenleaf Functions	185	**125**

imsi products

risC *Assembler Programming Tool*	80	**65**
TurboHALO *Specify Turbo C or Pascal*	80	**75**

komputerwerk products

Finally *BASIC Routines*	99	**85**
Finally Modules *New*	99	**85**
Finally XGraf *New*	99	**85**

RuleMaster2/PC by Radian

List $495 Ours $389

RuleMaster 2 is a set of software tools for the development of rule-based expert systems. Knowledge can be entered in the form you prefer, and the C or ForTran source code generators allow you to easily embed an expert system within another program. Complete English-language explanations assist you in understanding the line of reasoning used to arrive at a solution. The end-user interface is complete with pull-down menus and windows to enhance the ease of use of your expert system.

lattice products

Lattice C Compiler *ver 3.2 from Lattice*	500	**265**
with Library Source Code	900	**495**
C Cross Reference Generator	50	**37**
C-Food Smorgasbord *Function Library*	150	**95**
with Source Code	300	**179**
C-Sprite *Source Level Debugger*	175	**119**
Curses *Screen Manager*	125	**85**
with Source Code	250	**169**
dBC III	250	**159**
with Source Code	500	**318**
dBC III Plus	750	**594**
with Source Code	1500	**1184**
LMK *Make Facility*	195	**138**
RPG II Combo *All four items below*	1400	**1099**
RPG II Compiler *No Royalties*	750	**625**
Screen Design Aid Utility *for RPG II*	350	**309**
SEU *Source Entry Utility*	250	**199**
Sort/Merge	250	**199**
Text Management Utilities	120	**88**

meridian products

AdaVantage *GSA-validated*	795	**735**
with Optimizer *New*	995	**919**
AdaVantage Debugger *New*	495	**449**
AdaVantage DOS Environment Package	50	**47**
AdaVantage Utility Packages	50	**47**

metagraphics products

MetaWINDOW *No Royalties*	195	**138**
MetaWINDOW/PLUS	275	**195**
QuickWINDOW/C *for Microsoft QuickC*	95	**79**
TurboWINDOW/C *for Borland Turbo C*	95	**79**
TurboWINDOW/Pascal *for Borland Turbo Pascal*	95	**79**

microport products

DOSMerge286 *Specify 2-Users or Unlimited*	149	**129**
DOSMerge386 *2-Users*	395	**345**
DOSMerge386 *Unlimited Users*	495	**429**
System V/386 Combination	799	**669**
386 Runtime System	199	**169**
386 Software Development System	499	**429**
Text Preparation System	199	**169**
System V/AT Combination	549	**465**
AT Runtime System	199	**169**
AT Software Development System	249	**209**
Text Preparation System	199	**169**

microsoft products

Microsoft BASIC Compiler *for XENIX*	695	**439**
Microsoft BASIC Interpreter *for XENIX*	350	**219**
Microsoft C Compiler 5 *w/CodeView*	450	**285**
Microsoft COBOL Compiler *with COBOL Tools*	700	**439**
for XENIX	995	**639**
Microsoft Excel	495	**319**
Microsoft FORTRAN *Optimizing Compiler*	450	**285**
Microsoft FORTRAN *for XENIX*	695	**439**
Microsoft MACH 20	495	**329**
Microsoft Macro Assembler	150	**99**
Microsoft Mouse *with Paint & Mouse Menus*	150	**99**
with Microsoft Windows & Paint	200	**139**
with EasyCAD	175	**119**
Microsoft Pascal Compiler	300	**189**
for XENIX	695	**439**
Microsoft QuickBASIC	99	**66**
Microsoft QuickC	99	**66**
Microsoft Windows	99	**66**
Microsoft Windows 386	195	**129**
Microsoft Windows Development Kit	500	**299**
Microsoft Word	450	**285**
Microsoft Works	195	**129**

AdaVantage Compiler by Meridian Software

List $795 Ours $735
with Optimizer:
List $995 Ours$919

The AdaVantage Compiler is a fast, efficient, production-quality Ada programming environment fully validated by the U.S. Department of Defense. It generates native 8086 code in intel standard object format. Includes: compiler, linker, library mgmt. tools, support packages, runtime libraries, and a configuration tool. The optional Optimizer offers code improvements ranging from local optimizations to global subprogram removal.

mks products

MKS AWK	75	**65**
MKS RCS *Revision Control System*	189	**155**
MKS Toolkit *with MKS VI Editor*	139	**109**
MKS Trilogy *with AWK, CRYPT & Korn Shell*	119	**99**
MKS VI *Editor*	75	**65**

mmc ad system products

C Programmer's Toolbox I *New*	80	**69**
C Programmer's Toolbox II *New*	80	**69**
C Programmer's Toolbox Combo *New*	130	**115**

modula-2 language

LOGITECH Modula-2 Development System	249	**199**
Modula-2 Compiler Pack	99	**75**
Modula-2 Toolkit	169	**139**
LOGITECH Modula-2 Window Pkg	49	**39**
MODULA-2 *by Stony Brook*	195	**169**
with Utilities	345	**299**

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	**98**
with PLUS & PC Paintbrush	149	**119**
with PLUS & CAD Software	189	**153**
with PLUS & CAD & Paint	219	**179**
with PLUS & First Publisher	179	**139**
LOGIMOUSE C7 *with PLUS Package*	119	**98**
with PLUS & PC Paintbrush	149	**119**
with PLUS & CAD Software	189	**153**
with PLUS & CAD & Paint	219	**179**
with PLUS & First Publisher	179	**139**
LOGITECH HiREZ Mouse *for High-res Screens* *New*	149	**119**
LOGITECH Series 2 Mouse *for IBM PS/2* *New*	99	**79**
Microsoft Mouse *See Microsoft Section*		

novell development products

Btrieve *ISAM Mgr with No Royalties*	245	**184**
Xtrieve *Query Utility*	245	**184**
Report Option *for Xtrieve*	145	**99**
Btrieve/N *for Networks*	595	**454**
Xtrieve/N	595	**454**
Report Option/N *for Xtrieve/N*	345	**269**
XQL	795	**599**

peter norton products

Advanced Norton Utilities	150	**89**
Norton Commander	75	**55**
Norton Editor *New Version*	75	**55**
Norton Guides *Specify Language*	100	**65**
Norton Utilities	100	**59**

phoenix products

Pasm86 *Macro Assembler version 2.0*	195	**108**
Pdisk *Hard Disk & Backup Utility*	145	**99**
Pfantasy Pac *Phoenix Combo*	995	**595**
Pfinish *Execution Profiler*	395	**209**
Pfix86plus *Symbolic Debugger*	395	**209**
PforCe *Specify C Compiler*	395	**209**
PforCe++ *Specify C Compiler and C++*	395	**209**
Plink86plus *Overlay Linker*	495	**275**
Pmaker *Make Utility*	125	**78**
Pmate *Macro Text Editor*	195	**108**
Pre-C *Lint Utility*	295	**154**

pmi products

EmsStorage *New*	49	**45**
Graphix *New*	149	**119**
Macro2	89	**79**
ModBase	89	**79**
Repertoire	89	**74**
Repertoire/Btrieve Toolkit *New*	149	**119**

polytron products

PolyMake *UNIX-like Make Facility*	149	**129**
PolyShell *Special*	149	**105**
PolyXREF *Complete Cross Reference Utility*	219	**185**
PVCS Corporate *Version Control System*	395	**329**
PVCS Personal	149	**129**

pop-up reference guides

Peabody *by Copia Intl, Specify Language*	100	**89**
Resident Expert *by Santa Rita, Specify Lang*	CALL	**CALL**

sco products

XENIX System V for PS/2 *New*	CALL	**CALL**
XENIX System V 286	1295	**979**
Development System	595	**479**
Operating System	595	**479**
Text Processing Package	195	**144**
XENIX System V 386	1495	**1145**
Development System	695	**589**
Operating System	695	**589**
Text Processing Package	195	**144**

Personal Consultant Plus by Texas Instruments

List $2950 Ours $2589

Designed to take advantage of today's more powerful 286/386 DOS- based personal computers, Personal Consultant Plus utilizes extended knowledge representation features, and increased rule capacity. It allows you to use active values, meta-rules, browsing capabilities, frame descriptors, PC Scheme LISP functions, and an external program interface to C, Turbo Pascal, and more.

software garden products

Dan Bricklin's Demo II *New*	195	**179**
Dan Bricklin's Demo Program	75	**57**
Dan Bricklin's Demo Tutorial	50	**45**

software connection products

dB2c FILES *New*	199	**179**
dB2c TOOLKIT *New*	299	**249**
dB2c WINDOWS *New*	99	**89**

texas instruments

Arborist *Decision Tree Software*	595	**519**
PC Scheme *Lisp*	95	**77**
Personal Consultant Easy	495	**435**
Personal Consultant Images	495	**435**
Personal Consultant Online	995	**869**
Personal Consultant Plus	2950	**2589**
Personal Consultant Runtime	95	**84**

text editors

Brief & dBrief Combo *from Solution Systems*	275	**CALL**
Brief	195	**CALL**
dBrief *Customizes Brief for dBASE III*	95	**CALL**
Epsilon *Emacs-like editor by Lugaru*	195	**147**
Vedit Plus *by CompuView*	185	**128**
For XENIX *New*	285	**229**

turbopower products

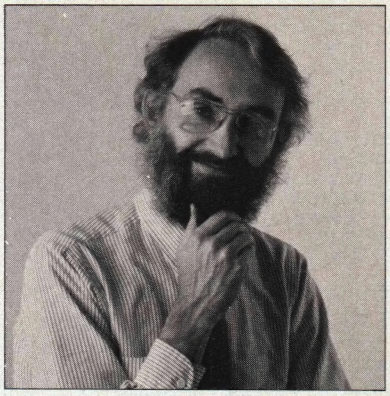
TDEBUG PLUS 4.0	45	**39**
with Source Code	90	**79**
Turbo Analyst 4.0 *New Version*	75	**59**
Turbo Professional 4.0	99	**79**

other products

ACTOR *by The Whitewater Group*	495	**419**
ApBasic *by CompTech Software* *New*	100	**79**
DocuMotion *by NWP-Intelligent Solutions* *New*	160	**139**
Heap Expander *by The Tool Makers* *New*	60	**55**
Interactive EASYFLOW *by Haventree*	150	**125**
JAM *by JYACC* *New*	750	**679**
MASTER*KEY *by Sharpe Systems* *New*	80	**69**
Opt-Tech Sort *by Opt-Tech Data Proc*	149	**99**
Pascal-2 *by Oregon Software*	295	**209**
pcAnywhere *by EKD Computer* *New*	99	**89**
Source Print *by Aldebaran Labs*	97	**74**
SourceTools *Make & Vers Cont by Oregon Soft*	595	**479**
Teamwork/PCSA *by Cadre Technologies*	995	**929**
TeleSwitch *by EKD Computer* *New*	289	**229**
TLIB *Version Control System by Burton*	100	**89**
Tmark *by Tangent Designs*	80	**69**
Tree Diagrammer *by Aldebaran Labs*	77	**59**
TurboGeometry *by Disk Software* *New*	100	**89**
XenoCopy-PC *by Xenosoft*	80	**69**
XenoFont *by Xenosoft* *New*	50	**45**

CALL for Additional Products

SWAINE'S FLAMES



Whan that April with his showres soote The droughte of March hath perced to the roote, And bathed every veine in swich licour, Of which virtue engendred is the flowr... Thanne longen folk to goon on pilgrimages, And palmeres for to seeken straunge strondes....

—Geoffrey Chaucer

Language evolves.

John McCarthy invented the LISP programming language almost 30 years ago, describing its overall design in an April 1960 *Communications of the ACM* paper. Since that ancient April, LISP has evolved in response to new environments: supercomputers, time-sharing systems, personal workstations.

During those decades, LISP maintained a solid reputation for being powerful, elegant, and inefficient. Remarkably inefficient. LISP code gobbled memory voraciously. LISP programs ran slowly, bogged down in function calls. That was the perception.

The perception was mainly accurate. Early compilers for LISP were crude and produced "simple and often ridiculous code on backing out of an execution-order treewalk of the program," according to Richard Gabriel, who wrote a book on benchmarking LISP implementations (*Performance and Evaluation of LISP Systems*, MIT Press, 1985).

Today, though, LISP compilers use all the tricks of the optimizing compilers for algorithmic languages. They convert costly function calls to in-line code, delay evaluation and rearrange the order of evaluation for efficiency, unwind loops, and do peephole optimization and constant subexpression elimination and interfunction optimization. If you think of LISP and FORTRAN, two ancient languages, as the archetypal symbol-processing and number-crunching languages, respectively, you might be surprised to learn that one modern LISP implementation, S-1 LISP, produces code for numeric computations that rivals FORTRAN code.

A LISP program is still likely to run slower and to use memory less efficiently than a comparable C program. But increased memory and processing power in current hardware and continuing improvements in LISP implementations make LISP more appealing for inclusion in developers' toolkits. This is good news because LISP's virtues—its extensibility, its expressive power, its facility in handling symbolic information—are impressive.

It's time somebody said it: spelling checkers are snake oil. In a recent editorial in another programming magazine, the editor (let's call him David) congratulated himself on the acquisition of a spelling checker, which he touchingly expected to cure the misspellings that have plagued his editorials. The selfsame editorial was, of course, plagued with the kinds of misspellings that spelling checkers don't check.

Spelling checkers do not, in fact, check spelling at all, and anyone who uses one for that purpose has been suckered by the snake-oil salesmen. If you want to know how a word is spelled, David, you look it up in a dictionary. The word lists supplied with spelling checkers are not dictionaries, whatever their purveyors may tell you, and can at best tell you that there exists a word with a certain spelling. Spelling checkers won't stop you from putting that damned apostrophe in possessive its. The commonest errors are the ones they are most likely to ignore.

Spelling checkers should really be called typo catchers because all they do is catch certain kinds of typing errors. Unfortunately, they don't even catch all of these. Consider the following mish-mash, which any spelling checker would accept unquestioningly: "Now is the tome fir all god men to sump this sneak oil sown the grain." What kind of error checking is it that regularly fails to detect the most common errors? Sneak oil indeed.

As you read this, it will have been nearly six months since Rob Dickerson made his pilgrimage from the Pacific Northwest to the straunge strondes of Scotts Valley, California. After some wintry blustering, Microsoft and Borland came to an agreement that put certain short-term constraints on Dickerson's activities as vice president of product management for Borland and also included a pact that the two companies would not recruit employees from one another's ranks for six months. Next month, when the moratorium on raiding ends and Nature priketh them in thir corages, will other folk longen to goon?

Michael Swaine

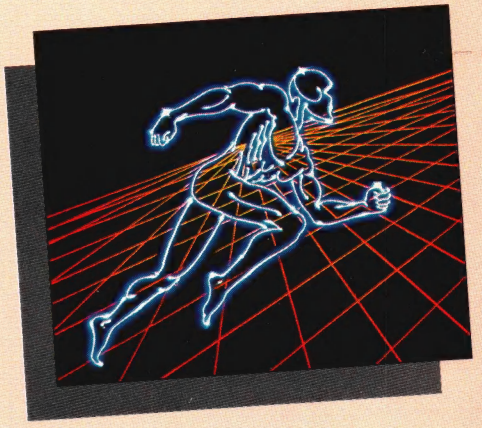
Michael Swaine
editor-in-chief

Introducing

FRONTRUNNER

New...for dBASE III PLUS Users! Fast...Resident...Powerful. FrontRunner offers all this and more!

- *CREATE MEMORY-RESIDENT dBASE III PLUS™ PROGRAMS – FrontRunner™ is the first memory-resident applications development tool to contain a large subset of dBASE III PLUS commands and allows you to distribute RunTime™ applications.*
- *dBASE III PLUS DATABASE AND INDEX FILE COMPATIBILITY – Allows you to use FrontRunner immediately.*
- *UNIQUE KEYBOARD FEATURE – Bind commands or entire programs to a single Hotkey for rapid execution from within other applications.*
- *PASTE COMMAND – This powerful command allows you to extract data from your dBASE III PLUS files and paste it into your spreadsheet or word processing application.*



Buy FrontRunner by June 30, 1988 and get a FrontRunner version of RunTime and an unlimited RunTime license for royalty-free applications. FrontRunner is not copy-protected and comes with a 30-day money-back guarantee. The suggested retail price is $195.

*See your local Ashton-Tate dealer now. For more information, or the name of the dealer nearest you, call (800) 437-4329, Ext. 556.**

**In Colorado, call (303) 799-4900, Ext. 556.*

ASHTON·TATE

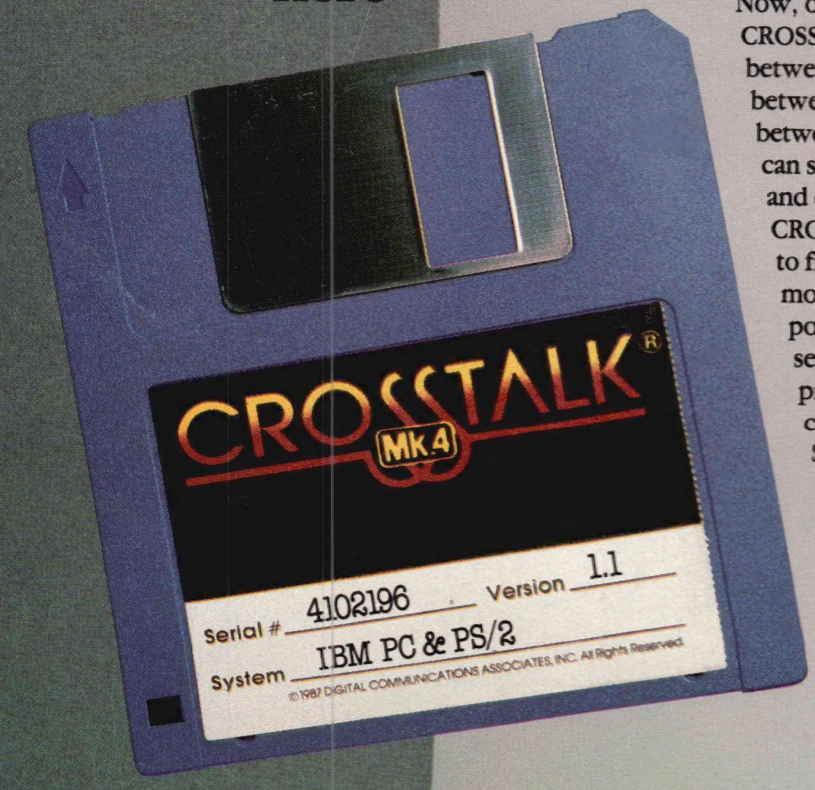
CIRCLE NO. 220 ON READER SERVICE CARD

IBM Spoken Here

and here

and here

and here



Whatever dialect of IBM you need to speak, CROSSTALK® Mk. 4 makes the connection.

Now, one program does the job that used to require several. CROSSTALK® Mk. 4 allows high-speed direct communications between PCs and minicomputers, or (with an IRMA™ board) between your PC and an IBM Mainframe, or (with Smart Alec™) between your PC and IBM System 3x's. If you like, CROSSTALK can support all of these sessions (and others) simultaneously, and display each session in its own window.

CROSSTALK Mk. 4 emulates all the terminals you're likely to find useful. That includes IBM 3101 (page and character modes), IBM 525x, IBM 529x, IBM 327x, as well as many popular async terminals like the DEC VT100 and VT220 series. CROSSTALK Mk. 4 includes the powerful CASL™ programming language, which allows you to automate communications applications quickly and easily.

So if you're used to thinking of CROSSTALK just to use with a modem, you're missing some important connections. Ask your dealer for details, or write:

dca Digital Communications Associates, Inc.
1000 Holcomb Woods Parkway / Roswell, Georgia 30076
1-800-241-6393

CROSSTALK® COMMUNICATIONS

CIRCLE NO. 221 ON READER SERVICE CARD